Spis treści rocznika

P. 1877/93
Biblioteka Główna

12 1993

informatyka

Wszystkim Czytelnikom i Autorom
życzymy miłych i spokojnych Świąt
oraz wiele radości i sukcesów
w nadchodzącym 1994 roku

Redakcja

WYDAWNICTWO SIGMA NOT

Cena 25 000 zł

informatyka

nr 12 1993
Grudzień
Rok wydania XXVIII

Miesięcznik
ISSN 0542-9951
INDEKS 36124

KOLEGIUM REDAKCYJNE:

mgr Jarosław DEMINET
mgr inż. Piotr FUGLEWICZ
mgr Teresa JABŁOŃSKA
(sekretarz redakcji)
Władysław KLEPACZ
(redaktor naczelny)
mgr inż. Jan RYŻKO
dr Zdzisław SZYJEWSKI

PRZEWODNICZĄCY
RADY PROGRAMOWEJ:

Prof. dr hab.
Juliusz Lech KULIKOWSKI

WYDAWCA:
Wydawnictwo Czasopism i Książek
Technicznych SIGMA NOT
Spółka z o.o.
ul. Ratuszowa 11
00-950 WARSZAWA
skrytka pocztowa 1004

Redakcja:
01-552 Warszawa,
Pl. Inwalidów 10, p. 104, 105
tel. 39-14-34

Materiałów nie zamówionych
redakcja nie zwraca

W sprawach ogłoszeń prosimy zwracać się bezpośrednio do Redakcji lub Działu Reklamy i Marketingu
00-950 Warszawa
ul. Mazowiecka 12
telefon: 27-43-66
telefaks: 26-80-16
teleks: 814877

W numerze:

Strona

SuperMemo as a new tool increasing the productivity of a programmer. A case study: programming in Object Windows – *Piotr Woźniak* 1
Logika teoretycznych urojeń – *Kazimierz Subieta* 5
Ośrodek obliczeniowy firmy ubezpieczeniowej w Kanadzie – *Janusz Gwiazda* 12
Modelowanie obiektów dynamicznych – *Stanisław Kędzierski* 16
Quo vadis, czyli ciąg dalszy o wyborze relacyjnej bazy danych – *Marek Miłosz* 20

Z kraju
II Konferencja EDI w Polsce 11
Fenomen OPTIMUSA 22
Konrad Zuse Profesorem Honorowym Uniwersytetu Szczecińskiego – *W.K.* 23

Roczny spis treści strony I–IV wewnątrz numeru

W najbliższych numerach:

Styczniowy numer INFORMATYKI 1994 roku w znacznej części zostanie poświęcony prezentacji materiałów wygłoszonych na I Krajowej Konferencji Elektronicznej Wymiany Danych – EDI, a mianowicie:

- **Marian Niedźwiedziński wprowadzi Czytelników w arkana EDI.**
- **Jan Durkiewicz przedstawi definicję EDI oraz zarys rozwoju tej techniki na świecie.**
- **Bogusław M. Hadyniak napisze o problemach zastosowań, uznając EDI jako innowację prowadzącą do zmian w sposobie funkcjonowania gospodarki i życia społecznego.**
- **Eugeniusz Danikiewicz scharakteryzuje międzynarodowy standard EDICACT (Elektroniczna wymiana danych dla administracji obrotu towarowego i transportu).**
- **Katarzyna Lange omówi podstawowe pojęcia związane z EDI na przykładzie wdrażania oprogramowania EDI w nowoczesnej firmie.**

Warunki prenumeraty na 1994 r.

Zamówienia na prenumeratę czasopism wydawanych przez Wydawnictwo SIGMA-NOT można składać w dowolnym terminie. Mogą one obejmować dowolny okres czasu, tzn. dotyczyć dowolnej liczby kolejnych zeszytów każdego czasopisma.

Zamawiający może otrzymywać zaprenumerowany przez siebie tytuł począwszy od następnego miesiąca po dokonaniu wpłaty. Zamówienia na zeszyty sprzed daty otrzymania wpłaty będą realizowane w miarę możliwości – z posiadanych zapasów magazynowych.

Warunkiem przyjęcia i realizacji zamówienia jest otrzymanie z banku potwierdzenia dokonania wpłaty przez prenumeratora. Dokument wpłaty jest równoznaczny ze złożeniem zamówienia.

Wpłat na prenumeratę można dokonywać na ogólnie dostępnych blankietach w Urzędach Pocztowych (przekazy pieniężne) lub Bankach (polecenie przelewu), przekazując środki pod adres:
Wydawnictwo SIGMA-NOT Spółka z o.o.
Zakład Kolportażu
00-716 Warszawa, skr. poczt. 1004
konto:
PBK S.A. III 0/Warszawa nr 370015-1573-139-11

Wpłaty na prenumeratę od marca br. przyjmują także wszystkie urzędy pocztowe nadawczo-odbiorcze oraz doręczyciele na terenie całego kraju

Na blankiecie wpłaty należy czytelnie podać nazwę zamawianego czasopisma, liczbę zamawianych egzemplarzy, okres prenumeraty oraz własny adres.
Na życzenie prenumeratora, zgłoszone np. telefonicznie, Zakład Kolportażu ul. Bartycka 20, 00-950 Warszawa, (telefony: 40-30-86, 40-35-89 oraz 40-00-21 wew. 249, 293, 299) wysyła specjalne blankiety zamówień wraz z aktualną listą tytułów i cennikiem czasopism.

Odbiorcy zagraniczni mogą otrzymywać czasopisma poprzez prenumeratę dewizową (wpłata dokonywana poza granicami Polski w dewizach, wg cennika dewizowego z cenami podanymi w dolarach amerykańskich) lub poprzez zamówioną w kraju prenumeratę ze zleceniem wysyłki za granicę (zamawiający podaje dokładny adres odbiorcy za granicą, dokonując równocześnie wpłaty w wysokości dwukrotnie wyższej niż cena normalnej prenumeraty krajowej).

Egzemplarze archiwalne (sprzedaż przelewowa lub za zaliczeniem pocztowym) można zamawiać pisemnie, kierując zamówienia pod adresem: Wydawnictwo SIGMA NOT, Spółka z o.o. Zakład Kolportażu, 00-716 Warszawa, ul. Bartycka 20, paw. B, tel. 40-37-31, natomiast za gotówkę można je nabyć w Klubie Prasy Technicznej w Warszawie ul. Mazowieckiej 12, tel. 26-80-17.

Istnieje możliwość zaprenumerowania 1 egz. czasopisma po cenie ulgowej przez indywidualnych członków stowarzyszeń naukowo-technicznych zrzeszonych w FSNT oraz przez uczniów zawodowych i studentów szkół wyższych. Blankiet wpłaty na prenumeratę ulgową musi być opatrzony na wszystkich odcinkach pieczęcią koła SNT lub szkoły.

W przypadku zmiany cen w okresie objętym prenumeratą Wydawnictwo zastrzega sobie prawo do wystąpienia o dopłatę różnicy cen oraz prawo do realizowania prenumeraty tylko w pełni opłaconej.
Cena jednego egzemplarza: normalna 35 000 zł, ulgowa 26 250 zł
Wartość prenumeraty w zł:
Normalna: kwartalna 105 000, półroczna 210 000, roczna 420 000
Ulgowa: kwartalna 78 750, półroczna 157 500, roczna 315 000.

Skład i druk: Drukarnia SIGMA NOT Sp. z o.o. z. 445/93

W ostatnim okresie coraz częściej napływały do nas sugestie, abyśmy zarówno w interesie krajowego środowiska, jak i podwyższenia rangi czasopisma jako organu profesjonalistów, zaczęli publikować teksty artykułów również w języku angielskim, który niewątpliwie stał się uniwersalnym językiem informatyków na całym świecie. Impulsem do wprowadzenia na nasze łamy tego rodzaju eksperymentu stał się referat ***Piotra Woźniaka*** *wygłoszony i opublikowany w tym języku podczas tegorocznej Szkoły PTI w Szczyrku. Ze względu na bardzo interesującą treść oraz wyjątkowo klarowny sposób jej prezentacji, postanowiliśmy opublikować ten tekst w całości jako sondażową próbę realizacji wspomnianych sugestii.*

Przy tej okazji zwracamy się do wszystkich Czytelników z gorącą prośbą o przekazywanie listownie lub telefonicznie (39-14-34) opinii na temat naszego eksperymentu.

Redakcja

POLITECHNIKA ŚLĄSKA
Biblioteka Główna
w Gliwicach
P. 1877/93

PIOTR WOŹNIAK
Poznań

SuperMemo as a new tool increasing the productivity of a programmer

A case study: programming in Object Windows

The paper provides general guidelines for increasing the productivity of a programmer by means of SuperMemo – a new speed-learning technique. The author has been working on a software development project involving programming in Object Windows (an application framework designed by Borland for Turbo Pascal for Windows). Simultaneously, a SuperMemo database has been being developed for training programmers working with Object Windows. Conclusions coming from the author's assignment are described and illustrated with examples.

PIOTR WOŹNIAK, born March 6, 1962, graduated from Adam Mickiewicz University in Poznań with a major in molecular biology (1985) and from the Technical University of Poznań with a major in programming techniques (1990). In the years 1982–1985, involved in experimental research on optimum repetition spacing, which led the to formulation of the SuperMemo method. In the years 1985–1990, within the framework of an individual course of studies under the supervision of Prof. Zbigniew Kierzkowski at the Technical University of Poznań, working on improving computer algorithms employed in computing optimum spacing of repetitions in SuperMemo. In 1991, cofounder of SuperMemo World, a company dedicated entirely to development and promotion of the applications of SuperMemo. In the years 1991–1993, working full-time in SuperMemo World in supervision of research and development.

Numerous studies on productivity in software engineering have shown that programming, which still combines elements of art and science, can be numbered among those few domains of engineering, where individual differences in productivity, often measured in the number of statements per day, vary greatly beyond ordinary expectation. The often quoted relationship is like 1 to 20 among programmers employed in the same project [1]. This paper presents a way to increasing the programmer's productivity by means of simple training techniques that can be introduced at a very little cost by software developers or even by individual programmers on their own.

There are two opposite trends affecting the complexity of problems encountered by an engineer using state-of-the-art technology in development project:

- introduction of new concepts and techniques at the receiving-end of new technologies increases the complexity of tasks facing the developer
- introduction of new tools and techniques at the level of the developer's workshop makes his attention shift from methodological aspects to conceptual aspects of the considered problem.

The subject of this paper is a new training technique. SuperMemo, which can be employed in offsetting the decrease of a programmer's productivity observed in association with increasing complexity of the development environment. The attention is focused on programming in Object Windows – an object-oriented application framework designed by Borland for Windows programmers privy to Pascal.

Introduction of Microsoft Windows version 3.0 in 1990 resulted in an increasing interest in graphic interface applications for DOS based personal computers. As the user of the application was provided with new standardized application interface tools such as pull-down menus, icons, dialog boxes, scroll bars, command buttons, etc., the programmer had to face new challenges posed by mastering close to 1000 functions of the Application Programming Interface, as well as not lower a number of constants, messages, structures, etc. Windows' Graphic Device Interface, and a number of other improvements at the level of the operating system, increased the productivity at the end-user level, and at the same time, rapidly increased the complexity of the development environment offered to the programmer. This situation lead to a popular saying that Windows programs are as good as they are because only the best programmers can complete them.

In consequence of a greater challenge facing the programmer, a number of application frameworks have been developed in order to counterbalance the growing complexity of programming in Windows. Increasingly, object-oriented approach has been used in similar situations over the past decade. One of the most popular application frameworks for Pascal programmers are Object Windows developed as a standard library for Turbo Pascal and Borland Pascal for Windows.

Optimally, development of new programming tools should gradually reduce the complexity of routine programming tasks in software engineering, and clearly separate the conceptual effort of the developer from details of the operating system, programming language and the programming techniques.

The optimum situation, however, seems to be ever more elusive as new requirements are imposed on software application with availability of new technologies, appearing periodically in all fields of computer and communications technology. Professional programming was, and continues to be the domain of the few, and success in dealing with complexity has its fundamental bearing on the overall effectiveness in software development projects.

The are four basis sources of information for a programmer working in software development:
- printed documentation and publications,
- on-line help facilities,
- peer-level and technical support consultation,
- programmer's knowledge acquired in the course of programming studies and programming practice.

There is a natural tendency to comprise the printed sources of information, as well as human-to-human exchange of information in increasingly intelligent computer help facilities. In other words, a clear split emerges between the two forms of storing knowledge: in a computer and in the human brain; phenomenon appearing more and more visibly in all kinds of scientific and engineering tasks.

The three related problems are:
- how to divide knowledge between computer and natural memory storage,
- how to represent knowledge in a computer knowledge base, and how to implement effective querying procedures,
- how to represent knowledge for human training, and how to effectively transfer in to the subject's brain.

In the presented paper, the first and the third problems are considered.

The obvious advantages of knowledge stored in the human brain are its high level of associativity and relatively short access and query time. Artificial Intelligence systems do not seem to closely approach humans in a number of highly complex intellectual tasks; therefore, there is still a special place for human brain and intelligence in software development, and time loss vs time gain analysis makes the basic criterion in choosing what what to store and what not to store in human memory.

In other words, time lost for training, later called time loss (TL) must offset the time lost in result of using an external reference. Time saved by not having to refer to external sources of knowledge will later be called time gain (TG).

Time loss and time gain, whose relationship determines which elements of knowledge should be mastered in the course of the training, can be computed individually on a one-by-one basis for each of the pieces of information. Those pieces considered in the analysis will later be called items, while the criterion $TG > TL$ will be called the memorization criterion.

Let us consider the time gain side of the memorization criterion. There are three major components here:
- reference time, i.e. time needed to look up the item in the external source of information, e.g. in a help facility,
- deadline penalty, i.e. extra time penalty coming from not having the item available before a certain deadline,
- non-association penalty, i.e. extra time penalty coming from not being able to produce associations, which form the basis of creative thinking.

Only the first component of the time gain can easily be estimated and practically employed in determining knowledge that should be transferred into the programmer's brain in the course of the training. The defficulty with the two other components is that the deadline penalty is a rare phenomenon in the practice of programming, while losses coming from non-association are very hard to estimate. Inability to form associations, a result of not having simultaneous access to unrelated components of the relevant knowledge, may involve a penalty that greatly surpasses the penalty coming from the need for extra knowledge references (first component of time gain).

The final form of the time gain equation in the practice of programming could be as follows:

$$TG = RefNo * RefT + NAP \quad (1)$$

where:
TG – time gain,
$RefNo$ – number of references needed for a given item,
$RefT$ – average reference time,
NAP – non-association penalty.

The time loss side of the memorization criterion can be computed with a high degree of accuracy, on condition that a systematic approach to the learning process is assumed.

SuperMemo is a learning technique, in which the spacing of repetitions of knowledge is optimized by means of an algorithm

proprietary to SuperMemo World. As it can be demonstrated, SuperMemo produces a repetition spacing that is close to the optimum spacing for a selected level of knowledge retention.

From the optimum spacing of repetition, and from practical measurements of mean repetition times, the time loss per item can be computed for a selected period of time and a selected level of knowledge retention.

The accurate formula for time loss could like this:

$$TL = RepNo * RepT \quad (2)$$

where:
TL – time loss in the training period,
RepNo – number of repetitions in the training period,
RepT – average repetition time.

Practice shows that knowledge retention of about 95% provides the best balance in the retention vs acquisition rate trade-off in learning material useful from the programmer's standpoint (this level of knowledge retention is also recommended in majority of practical applications of SuperMemo [4].

It can also be shown that the knowledge of the highest degree of applicability in the practice of programming has extraordinarily short lifetime. Unlike in sciences or medicine, the knowledge of a programmer must constantly be turned over, with a large proportion of items losing their applicability is a period of time as short as 1–3 years.

As an illustration of item attrition in databases pertaining to the programming knowledge, let us consider an example of a SuperMemo database which was developed by the author with the view to learning Pascal programming, starting with Turbo Pascal 3.0 and ending with Borland Pascal 7.0 for Windows.

Year	Language	BP7	TPW	TP6	TP4	TP3	Total
1987	TP3	–	–	–	–	800	800
1989	TP4	–	–	–	1000	600	1000
1991	TP6	–	–	1500	1000	600	1500
1992	TPW	–	3500	1100	700	400	3500
1993	BP7	4000	3100	1100	700	40	4000

Where:
Year – the year in which database analysis was done,
Language – the Pascal superset in question (Turbo Pascal 3.0, Turbo Pascal 4.0, Turbo Pascal 6.0, Turbo Pascal for Windows 1.0 and Borland Pascal 7.0 for Windows respectively),
BP7, TPW, TP6, TP4, TP3 – number of items coming from the particular superset in the entire SuperMemo database.

The above figures illustrate (1) a substantial growth in the number of items coming from the increase in the programming system complexity resulting from the introduction of Turbo Pascal 1.0 for Windows, as well as (2) a high attrition rate of items in the database in accompaniment of employing new language elements, tools and techniques to Turbo Pascal.

The average number of repetitions per item is about 5–6 in a well-structured SuperMemo database. As the average repetition time is from 5 to 7 seconds, the time loss per item is about 30–40 seconds in the considered period of time.

From equations (1) and (2) we can easily obtain a simple formula for estimating the memorization criterion:

$$RefNo * RefT + NAP > RepNo * RepT$$

Considering that in the period of 2 years RepNo = 6 and RepT = 6 we can reduce the inequality to the following form:

$$RefNo * RefT + NAP > 36\,[seconds] \quad (3)$$

we will use the above formula for the memorization criterion in the analysis of exemplary items to verify their suitability for memorization as opposed to being left for further reference.

In the course of developing a SuperMemo database on programming in Object Windows, the author has identified the following six dominant aspects that should be subject of training by means of SuperMemo:

- Foundations of computing sciences: algorithms, data structures, computational complexity, probability calculus, operation research, numerical methods, software engineering, statistics, etc.
- Elements of the operating system and the assembly language (in the considered case: DOS, Windows and the 80x86).
- Elementary terminology and concepts of the development environment (for example, in Windows: client area, control menu, system key, modal vs modeless dialog, child vs popup window, logical vs device co-ordinates, interface object vs interface element, virtual method table, polymorphism, etc.)
- Programming tools (debugger, profiler, resource workshop, help compiler, etc.)
- Elements of the language; syntax and use: procedures, functions, variables, constants, objects and their hierarchy, fields, methods, messages, run-time errors (but not compiler errors), etc.
- Practical tips, observations, and experimental findings not included in the documentation (these should be collected by the programmer in the course of the programming practice). For example, compiler bugs, freaks of the operating system, frequently encountered traps (e.g. using more than five device contexts in the GDI), etc.

Let us consider the memorization criterion for exemplary items used to learn programming in Object Windows, and belonging to the fifth knowledge domain, that of elements of the language. All the examples are taken from the database developed by the author. Letter Q and A are used to denote questions and answers.

Example 1:

Q: What is the name of the pointer to the main window (defined by TApplication)?
A: MainWindow.

In Object Windows, the application instance is defined as an object whose type is TApplication. One of the fields in TApplication is a pointer to the main application window, MainWindow.

The name of the pointer can easily be found in the Borland Pascal on-line help. The reference time (RefT) can be estimated to be no less than 20 seconds assuming the knowledge of the name TApplication, or at least 40 seconds otherwise (486 class computers). The number of references (RefNo) depends greatly on the programmer's needs and his or her programming style. Main Window is used in the method InitMainWindow, which must specified for every Object Windows program. If the programmer works on a single project, he is more likely to reference MainWindow as the parent of other windows or dialog boxes. RefNo can be estimated as varying from 0 to 100 for the period of two years. The item in question asks for an identifier, and as such involves a very low nonassociation

penalty (NAP). A conservative substitution to the memorization criterion could therefore look as follows: TefNo*20 > 36, which justifies introducing the analysed item into a SuperMemo database as soon as the expected number of references is greater than one! It is important to notice, that a large number of references (above 20) may result in memorization of the item in question without the help of SuperMemo. However, memorization criterion does not lose its validity as long as the first reference does not result in a memory engram that can be sustained until the need for the next reference arises.

Example 2:

Q: What are the two stages of MakeWindow?
A: (1) Checking for memory conditions (ValidWindow).
(2) Creating the window (Create).

In Object Windows, MakeWindow is a method of TApplication, which takes a pointer to a window as its parameter, and is responsible for creating the relevant interface element. The knowledge of the stages of MakeWindow is not needed by the programmer in the coding process therefore RefNo can safely be put to zero. However, because of its educational aspect, and because of possibility of using the method Create, which is not recommended, the non-association penalty (NAP) can safely be set as greater than 36 seconds needed to satisfy the memorization criterion.

Example 3:

Q: What is HWindow?
A: Handle to the interface element of a given object

HWindow is used by all objects TWindowsObject and its descendants to store the handed to the corresponding interface element, or to store zero if the interface element has been destroyed or not created at all. Interestingly, this item does not satisfy the memorization criterion. Passive recognition of the identifier HWindow is needed in programming Object Windows; however, because of its frequent use and intuitiveness, the recognition can easily be achieved without extraneous training (i.e. any training different than active use of the identifier in the programming practice). The reason for placing this item in the database is different, and comes, from the principles of SuperMemo, which demand that multifaceted approach to learning be taken. What the programmer has to remember and what satisfies the memorization criterion is the active recall of the name HWindow. However, adding a symmetrical item which, in a sense, duplicates the memorized piece of knowledge and asks for its passive recognition, has been proved to reduce the overall workload, i.e. the time loss side of the memorization criterion (TL). This happens by reducing the difficulty of the items in question; the fact reflected by a lower number of repetitions per item, e.g. 6 repetitions in two years instead of 10 repetitions, and by a shorter repetition time, e.g. 3 seconds instead of 5 seconds (the overall difference here would be as between 2 items * 6 repetitions * 3 seconds = 36 seconds, and 1 item * 10 repetitions * 5 seconds = 50 seconds).

Example 4:

Q: What object types define HWindow?
A: TWindowsObject (and all the descendants).

Though again, the estimated reference number (RefNo) is zero, this item has been introduced into the database in result of a combination between the non-association penalty and the multifaceted approach stipulated by SuperMemo.

Example 5:

Q: What are the parameters of GetText in edit controls?
A: (1) Retrieved text and (2) its maximum size.

Edit controls are standard Windows interface objects used in editing null-terminated strings. The method GetText defined by the object type TEdit, a descendant of TWindowsObject in Object Windows, retrieves the text from an edit control, and requires the maximum size of the retrieved text as the second parameter. This item is a typical example of a language item that fully satisfies the memorization criterion. Here, the reference time should rather be called a recompilation penalty time, because the main hitch in using GetText comes from the fact that it returns the size of the retrieved text, not the text itself. The mistaken reference of GetText will easily be discovered at compile time as a type mismatch. The penalty involved could span from 10 to 45 seconds depending on the size of the compiled code. The reference number (RefNo) can range from 10–100, and a conservative substitution to the memorization criterion could look as follows: RefNo*10 > 36 [seconds], which implies that four recompilations would justify introducing the item into the database.

Example 6:

Q: What are the descendant types of TStatic?
A: TEdit.

Static controls, encapsulated in Object Windows in the object type TStatic, are standard Windows interface objects which define static texts such as edit control labels in a dialog box. The above item is an example of benefits coming from reasonable knowledge structuring in learning highly associative subject domains. From the assertion implied here, and from the understanding of inheritance in object-oriented programming, the programmer can infer a number of properties of objects typed as TEdit. For example, by specifying items needed for understanding the SetText method of TStatic, the programmer does not have to duplicate the same questions for TEdit. Obviously, the multifaceted approach principle could be raised to justify such redundancy; however, some extra components of the memorization criterion would have to be added in order to balance the trade-off between time gain coming from multifacetedness and time loss coming from the extra workload.

Let us have a short look at the overall workload involved in the training based on SuperMemo. In the course of the project which started 401 days before writing this paper, the analysed SuperMemo database has grown to comprise 3605 items. After 401 days, the average number of repetitions per day was 47, the average number of repetitions per item was 4.5, the average repetition time was 7.4 seconds per item, and the average daily workload was 5 minutes and 44 seconds (excluding time needed to extract items from the documentation and to put them into the database). On average, 1 page of documentation produced 3–4 items, which indicates that the database makes up an equivalent of 900 pages of printed reference (Borland Pascal documentation, DOS and Windows manuals, and the WINDOWS programming reference total over 7000 pages altogether).

Basing on the experience gained in the course of compiling the discussed database, the following exemplary 8-hour working schedule could be proposed for a software development project, in which new tools or a new environment are to be employed:

dokończenie na s. 11

KAZIMIERZ SUBIETA
Instytut Podstaw Informatyki PAN
Warszawa

Logika teoretycznych urojeń

Po kilku latach spędzonych za granicą odwiedziłem sklepy komputerowe w poszukiwaniu sprzętu, oprogramowania i literatury niezbędnej do pracy w nieco nowym dla mnie świecie PC. Wśród różnorodnych „Turbo", „Windows", „Dosów" i „Wordów" odkryłem także książkę autorstwa G. Mirkowskiej i A. Salwickiego „Logika algorytmiczna dla programistów" (WNT 1992). Dla programistów! Zajmując od lat pozycję pomiędzy światem akademickim i światem komercyjnym dawno wyrobiłem sobie pogląd na temat logiki algorytmicznej i jej stosunku do praktyki; tytuł i miejsce sprzedaży książki pozostawały z tym poglądem w wyraźnej sprzeczności. Zaintrygowany, wypożyczyłem książkę i poświęciłem jej kilka wieczorów. Zwykle lektura podobnych dzieł nie wzbudza we mnie żadnych emocji. Tym razem jednak książka była zbyt prowokująca. Uznałem, że stanowi ona dobry pretekst do podjęcia dyskusji nad rolą zmatematyzowanych teorii w dziedzinie informatyki, oraz dostatecznie reprezentatywny przykład ilustrujący niektóre moje poglądy i tezy. Nie jest to tylko sprawa moich poglądów: uważam, że normalne funkcjonowanie wielu ludzi, orientujących swoje badania na informatykę rzeczywistą, będzie w niektórych środowiskach nadal utrudnione i owocujące w nieporozumienia i konflikty, o ile wszyscy pozostaną obojętni wobec postawy naukowej i twórczości, których rezultatem jest ta książka.

Zanim przejdę do ocen, kilka uwag generalnych. A. Einstein słusznie powiedział, że nie ma nic bardziej praktycznego, niż dobra teoria. W fizyce stosunek teorii i rzeczywistości jest klarowny i nie podlegający dyskusji. Teoria jest całkowicie podporządkowana doświadczeniu: najwspanialsza, pięknie zmatematyzowana teoria zamienia się w makulaturę, o ile przeczy jej jedno doświadczenie. Teorie fizyczne są abstrakcjami opisującymi świat obiektywny, niezależny od ludzkiej woli. Stąd wartość teorii fizycznej jest wyznaczona tym, na ile precyzyjnie pozwala ona wnioskować o stanie i zjawiskach tego świata.

Moim zdaniem, ten stosunek pomiędzy teorią i rzeczywistością nie przenosi się na informatykę (generalnie, nie przenosi się na nauki inżynieryjne, do których informatyka należy). Świat informatyczny nie istniał obiektywnie: został utworzony przez ludzi, zaś jego przyszły kształt zależy wyłącznie od rezultatów ludzkiej twórczości. Rola teorii informatycznych mogłaby polegać na opisie, analizie i syntezie jej uformowanej rzeczywistości i w tej roli przypominałyby one teorie fizyczne. Do takich teorii można zaliczyć liczne zmatematyzowane idealizacje (nazywane w fizyce „fikcjami"). Przykładem (pierwszym z brzegu) jest rachunek lambda, używany za narzędzie do badania funkcjonalnych języków programowania; ale przecież wiadomo, że języków *idealnie* funkcjonujących (bez pojęcia stanu) w praktyce nie ma i być nie może. Podobnie z gramatykami bezkontekstowymi: stanowią one podstawę implementacji wielu języków, ale każdy z nich wprowadza dodatkowe ograniczenia syntaktyczne (np. statyczną zgodność identyfikatorów i typów), które czynią z niego język kontekstowy. Takie teorie przydają się jako wzorzec metodologiczno-dydaktyczny: objaśniają zjawiska informatyczne na poziomie koncepcyjnym, ale przez liczne i konieczne odstępstwa rzeczywistości od ideału dość często nie stosują się do sytuacji konkretnych, wymagają niekiedy dramatycznych korekt, lub ich zastosowanie do sytuacji konkretnych jest żmudne i pozbawione pragmatycznych celów (o ile wyłączymy z tych celów kariery naukowe).

Chociaż zmatematyzowane idealizacje rzeczywistości można uznać za główny motyw napędzający rozwój informatycznych teorii, większość z nich do tej klasy nie należy. Rzeczywistość informatyczna, budowana przez nawarstwianie celów, interesów, pomysłów, wynalazków – nieregularna, przypadkowa i arbitralna – nie jest szczególnym przedmiotem zainteresowań zmatematyzowanych teorii. Ich głównym celem jest kształtowanie przyszłości. Teorie, opierając się na faktach z istniejącej rzeczywistości i znajomości jej wad, postulują nową, lepszą rzeczywistość. W tej roli teorie przestają być teoriami sensu stricto, a zamieniają się w ideologie ustalające język, pojęcia, doktryny i reguły dotyczące przyszłych tworów informatycznych. Przykładowo, model relacyjny baz danych, dedukcyjne bazy danych, programowanie w języku logiki, itd. mają charakter doktryn ideologicznych, które pozostają doktrynami również wtedy, gdy są zapisane i badane za pomocą aparatu matematycznego.

W ten sposób ustaliłem klasę, do której należy teoria prezentowana w przedmiotowej książce. Jest to zmatematyzowana ideologia, która proponuje nową jakość w dziedzinie programowania. Zdaniem Autorów, wadą obecnych technologii programowania jest to, że nie sposób jest ustalić przed wykonaniem programu, czy program jest „poprawny" (tj. czy jest zgodny ze swoją specyfikacją) i czy nie zapętli się przy jakiejś kombinacji danych wejściowych. Aby złu zaradzić, Autorzy proponują wyspecjalizowaną logikę algorytmiczną, stanowiącą rozszerzenie logiki klasycznej, za pomocą której można będzie precyzyjnie wyspecyfikować, co program robi. Formuły logiczne stanowiące tę specyfikację można następnie użyć w celu przeprowadzenia dowodu poprawności programu.

Mało zorientowany Czytelnik zapewne nie dostrzega konsekwencji tej propozycji. Postaram się je uświadomić. Podana technologia ma sens wyłącznie dla dużych programów, ale (ze zrozumiałych względów) przykłady w książce nie są dłuższe niż kilka linijek. Dla tych programów długość specyfikacji w logice algorytmicznej jest porównywalna z długością programu (a należy oczekiwać, że dla mało „gadatliwych" języków programowania, takich jak np. LISP lub Prolog, długość specyfikacji w logice algorytmicznej może przerosnąć długość programu). Dla dużych systemów CAD, kompilatorów, systemów zarządzania bazami danych, itp. długość programów mierzy się w setkach tysięcy lub milionach linii. Przykładowo, system POSTGRES liczy blisko miliona linii w języku C. Książka o logice algorytmicznej dla programistów liczy około 300 stron,

po ok. 35 linii na stronie. Bądźmy liberalni przyjmując, że specyfikacja będzie pięciokrotnie krótsza od programu. Do wyspecyfikowania systemu POSTGRES należałoby więc wyprodukować ok. 20 takich tomów, zawierających wyłącznie formuły logiki algorytmicznej!

Teraz zapewne każdy Czytelnik dostrzega wady całego pomysłu. Kto podejmie trud sporządzenia specyfikacji o takiej długości? Jak zapewnić jej bezbłędność, jeżeli uważny człowiek robi około dwóch błędów na jednej stronie, nie licząc błędów semantycznych i koncepcyjnych. Kto sprawdzi, że ta specyfikacja jest zgodna z tym, co chciał klient zamawiający program, tym bardziej, że najczęściej klient sam nie wie, czego dokładnie chce. Jak wyspecyfikować scenariusz i środki techniczne (formy, okna, menu, grafika, etc.) interakcyjnego interfejsu użytkownika? Jak wyspecyfikować struktury danych złożone z setek typów obiektów powiązanych w sieć różnorodnych zachodzących między nimi związków? Gdzie szukać matematyków, którzy z tysięcy formuł logicznych wyprowadzą twierdzenie o tym, że program jest poprawny[1]. Jak w tej technologii wyobrazić sobie rozliczne modyfikacje, przeróbki i doróbki, które są codziennym chlebem projektantów i programistów? Gdzie jest tu miejsce na różne poziomy abstrakcji, niezbędne do mentalnego opanowania specyfikacji bardzo dużych programów? Jak mogłoby wyglądać uzasadnienie tej metody od strony ekonomicznej, ergonomicznej i organizacyjnej?

Pytania są retoryczne: książka nie udziela na nie żadnych odpowiedzi. „Specyfikacja" oznacza tam rodzaj podwójnego programowania, w normalnym języku i w języku logiki algorytmicznej, języku lakonicznych symboli i kodowanych znaczeń, a więc w języku trudnym do opanowania i prowadzącym do błędów. Zresztą, w książce nie ma wzmianki o percepcji tego języka przez przeciętnego projektanta lub programistę; to nie stanowi przedmiotu zainteresowań teorii. Co za tym idzie, trudno dać wiarę autorom, że logika algorytmiczna *...jest lepiej poznana i ma więcej udokumentowanych zastosowań...* (bez wspominania, o jakie zastosowania chodzi), gdyż zdanie to sugeruje znacznie więcej, niż można od tej logiki oczekiwać. Jakież to systemy zostały wyspecyfikowane za pomocą logiki algorytmicznej, i czy aby na pewno ich programy były dłuższe niż 100 linii? Autorzy powołują się na przykład z dziedziny mechaniki, przeprowadzając analogię między programem i mostem, który może się zawalić, jeżeli zostaną zaniedbane odpowiednie obliczenia. W odróżnieniu jednak od konstruktorów mostów, którzy muszą starannie używać zaawansowanej matematyki lub wyspecjalizowanych matematycznych narzędzi, w pracy programisty nietrywialna matematyka nie daje się wykorzystać (o tym dalej). Nie jest to wada, jak sugerują Autorzy: istnieje wiele ważnych dziedzin, w których matematyka nie daje się wykorzystać (np. w chirurgii serca), wobec czego korzysta się z innych metod. Twierdzenie Autorów, że *...praca twórców oprogramowania nie doczekała się odpowiednich dla tego zawodu pomocy programistycznych...*, jest jedną z oznak niekompetencji autorów: sugeruje, że nie słyszeli oni np. o CASE (*Computer Aided Software Engineering*) i nie są świadomi stanu dużych dziedzin zwanych *debugging, rapid software prototyping, executable specifications, contex-sensitive program editing, make files, versioning* etc. Dodam, że sama dziedzina CASE zaowocowała dziesiątkami metodologii i systemów komputerowych, np. CCA DB Designer, ERwin firmy Bachman's, IEW firmy Knowledge Ware, DACOM firmy Janus, Oracle CASE, STRADIS i ProKitWORKBENCH firmy McDonnell Douglas, MetaSoftware, Merise, DATAID, system J. Martina, Gambit i wiele innych.

[1] Znam ewentualną odpowiedź – automatyczne dowodzenie twierdzeń, ale ono nie zechce działać nawet dla setek aksjomatów, nie mówiąc o tysiącach.

Z drugiej strony książka jest matematycznie kompletna i prezentuje oryginalną matematyczną teorię. Stajemy zatem przed dylematem, jak ocenić tę książkę od strony naukowej? Co to jest „nauka" w dziedzinie informatyki i czy książka jest naruszeniem norm obowiązujących w nauce? Stefan Amsterdamski w znakomitym eseju „Czy istnieje etos uczonych" (Wiedza i Życie, 1, 1992) skłania się do tezy, że definicja nauki grzęźnie w dywagacjach metodologiczno-pragmatycznych. Idealistycznie, pojęcie nauki powinno być wyznaczone nie użytecznym celem, lecz metodą. Wymuszanie z zewnątrz celów pragmatycznych może być kwestionowane od strony etycznego ideału, ale jest nieodzowną normą. Istotnie liczba wszystkich teorii naukowych, które można stworzyć, znacznie przekracza liczbę tych teorii, które stanowią użyteczną wiedzę. Stąd prawie każda nauka ulegnie degeneracji, o ile nie będzie weryfikowana przez użyteczność. Dążenie nauki do autonomii, której wyznacznikiem jest swoboda w wyborze problemów i metod, kłóci się z motywacjami społeczeństw, które utrzymując naukę oczekują od niej nie przypadkowych teorii, a takich efektów, które zmienią ich życie na lepsze. Zatem pojęcie nauki musi włączać elementy pragmatyczne, i na tej płaszczyźnie zachodzi istotna różnica między matematyką i informatyką. Naukowiec-matematyk nie musi dbać o użyteczność swojej twórczości, wystarczają racje filozoficzno-estetyczne i akceptacja kręgu matematyków. Dla naukowca-informatyka użyteczność jest podstawą; bezużyteczna teoria jest pseudo-teorią, badania nie owocujące w użyteczne wyniki są pseudonauką.

W ten sposób nadużycie, którego dopuścili się Autorzy i wydawcy książki wobec środowiska informatyków, można sprowadzić do tytułu zawierającego określenie „dla programistów", oraz fragmentów sugerujących bezpośrednie zastosowanie praktyczne. Bez tego tytułu i tych fragmentów książka mogłaby być uznana za jeszcze jedną matematyczną teorię, której wartość leży w sferze intelektualno-estetycznej – i nic poza tym. To oczywiście zmieniłoby inne kwalifikacje: nie jest wykluczone, że powstałyby wtedy trudności ze znalezieniem dla niej zarówno wydawcy jak i czytelnika. Można przypuszczać, że sztucznie doczepione do książki racje pragmatyczne powstały w wyniku nacisków zewnętrznych lub wewnętrznego odczucia potrzeby uatrakcyjnienia jej przedmiotu.

Nie będę starał się ustalić, czy mimikra zastosowana przez Autorów była świadomą próbą wprowadzenia w błąd producentów i odbiorców książki, czy jest to skutek nieświadomości stanu i osiągnięć informatyki praktycznej. Przypuszczałbym raczej to drugie. Od lat zamknięci w samo-podtrzymującym się, izolowanym świecie niepraktycznych teorii, Autorzy prawdopodobnie nie zdają sobie nawet sprawy z tego, że logika algorytmiczna jest i na zawsze pozostanie dla programistów „zbędna i niepotrzebna" (od czego się wstępnie odżegnują). Tego faktu nie jest w stanie obalić ani powołanie się na autorytety tzw. teorii programowania, ani też retoryka naiwnych i tendencyjnych argumentów, którymi Autorzy raczą Czytelnika we wstępie książki. Zresztą, większość osób działających w obszarze informatyki rzeczywistej nie da się już na tę retorykę nabrać. Ostrzeżenie można kierować tylko do studentów i początkujących informatyków, którzy jeszcze nie wiedzą, że fachowości, kompetencji i rzetelności zawodowej w dziedzinie informatyki nie da się zastąpić zmatematyzowaną pianą.

W tym miejscu chciałbym, aby Czytelnik nie odkładał jeszcze tego tekstu, uznając, że nie warto poświęcać więcej czasu sprawie, której wady są tak ewidentne; w końcu nie jest to ani pierwsza, ani (prawdopodobnie) ostatnia nieudana książka z dziedziny informatyki. Chciałbym zwrócić uwagę, że powstała ona w czołowym polskim ośrodku naukowym, zaś Autorzy są znanymi teoretykami informatyki o znacznym dorobku i auto-

rytecie. Logika algorytmiczna stanowiła przedmiot naukowych zainteresowań osób należących do polskiej czołówki informatyki akademickiej; stanowiła też (stanowi?) przedmiot nauczania na uniwersytecie. Pojawienie się tej pozycji na polskim rynku nie jest więc przypadkiem; jest to skutek generalnej patologii informatyki akademickiej, rozwijającej się przez wiele lat. Tę patologię praktycy określają różnymi pejoratywnymi zwrotami; unikając ich, określę ją jako *kontynuowanie sformalizowanych badań pomimo wyczerpania się potencjału matematyki jako narzędzia badawczego w wielu działach informatyki*, w szczególności w dziedzinach programowania, inżynierii oprogramowania i baz danych. Dalszą część artykułu poświęcę temu zagadnieniu.

W istocie, logika algorytmiczna jest kontynuacją paradygmatów naukowych, którymi żył świat informatyki akademickiej przed laty: teoria algorytmów i funkcji rekurencyjnych, maszyny Turinga, problem stopu, Hoare, Dijkstra, etc. Jest dinozaurem, który niespodziewanie wykluł się w innej epoce i innym klimacie. Dzisiaj te problemy niewiele kogo obchodzą. Kompletność obliczeniową w sensie Turinga osiąga się za pomocą banalnych środków i nie mówi ona nic istotnego o języku programowania, pojęcie algorytmu stało się drugorzędne i wspomina o nim bardziej w kontekście mikrotechnicznym niż w kontekście architektury i własności funkcjonalnych systemów, bardzo istotna własność algorytmów – efektywność – nie poddaje się teoriom[2], aksjomatyczne definicje semantycznych własności programów okazały się bezużyteczne itd. Świat użytecznych zmatematyzowanych teorii skurczył się, stał się niespełnioną nadzieją. Jest zrozumiałe, że epigoni przebrzmiałych formalizmów starają się ulokować je w obecnej rzeczywistości, nadać im aktualny kształt uzasadniający celowość ich działalności. Jeżeli prawie cały naukowy życiorys i kariera były związane z tymi formalizmami, jakże teraz je odrzucić?

Temat roli matematyki w informatyce i tzw. *matematycznych podstawach informatyki* zajmuje ostatnio sporo miejsca w literaturze i wraca jak bumerang w różnych dyskusjach. Jest charakterystyczne, i spotkałem się z tym zarówno w kraju jak i za granicą, że temat ten podnoszą dość często sami teoretycy informatyki, tak jak gdyby mieli jakieś poczucie winy lub kompleksy względem postępów informatyki praktycznej. Po prostu życie nie zastosowało się do przewidywań teoretyków wróżących wielką przyszłość metod matematycznych w informatyce.[3] To, co wydawało się ważne dla małych programów, którymi zajmowali się matematycy, okazało się nieistotne dla bardzo dużych programów. Fizyka atomowa nie nadaje się do badania konstrukcji samochodów, mimo, że samochód składa się z atomów, i jest to analogia z tym, co zdarzyło się w informatyce w ciągu ostatnich 30. lat. Twórczość informatyczna rozciągnęła się na ogromny obszar wyznaczony przez osiągnięcia w zakresie technologii komputerowych i dziedzin zastosowań, i ograniczony przez czynniki ekonomiczne, społeczne i ludzkie. Szczególnie te ostatnie, wymienię tu: zdolności, instynkty, organy percepcji i ekspresji, pojemność i trwałość pamięci, skłonność do błędów i nadużyć, lenistwo, bezmyślność, oporność na modyfikacje, wiara w autorytety, uleganie modom etc., etc., stwarzają istotną barierę dla stosowalności sformalizowanych teorii.

Syndromem nowej epoki w informatyce jest złożoność i wieloaspektowość produktów informatycznych, dostosowanie ich do aktualnych osiągnięć w zakresie sprzętu i oprogramowania, maksymalizacja ich funkcjonalności oraz ogromna liczba kompromisów powodowanych koniecznością uwzględnienia czynników ludzkich. Powstaje więc pytanie, jaka w tej epoce ma być rola teorii, w tym zmatematyzowanych teorii, i czy w tej nowej epoce można znaleźć sensowne miejsce dla takich teorii, jak logika algorytmiczna.

Obszar oddziaływania teorii informatycznych można podzielić na strefę *produkcji* informatycznej (wytwarzanie oprogramowania i innych produktów informatyki przy zastosowaniu dobrze opanowanych technologii), strefę szeroko rozumianej *wynalazczości* (tworzenie nowych jakości w zakresie sprzętu, oprogramowania, zastosowań, metodologii projektowania, niezawodności itd.), oraz strefę kształcenia informatyków i użytkowników.

Jest bardzo trudno uzasadnić oddziaływanie zmatematyzowanych teorii w strefie produkcji informatycznej; programista w mniejszym stopniu potrzebuje matematyki niż ekonomista lub konstruktor mostów. W moim życiorysie inżynierskim znalazło się miejsce na zastosowanie szeregów Fourier'a do analizy składowych harmonicznych sygnału, liczb zespolonych do analizy impedancji układów elektronicznych, rachunku operatorowego do obliczeń przebiegów nieustalonych, metod opartych na algebrze Boole'a do minimalizacji układów logicznych, itp. Wspaniała matematyka, którą musi znać kompetentny inżynier-elektronik. Podobne zmatematyzowane narzędzia praktycznie nie istnieją w pracy projektanta i programisty systemów, zaś oddziaływanie intelektualne metod o matematycznym rodowodzie jest marginesem; co najwyżej, tak jak np. w gramatykach LL(1), sprowadza się do prostych, inżynierskich intuicji. Dodam, że będąc już doktorem informatyki spędziłem parę lat jako zwyczajny projektant i programista w niemieckiej firmie softwerowej.

Moim zdaniem, nie istnieją również przekonywujące argumenty na rzecz oddziaływania zmatematyzowanych teorii w strefie wynalazczości. Istnieje koncepcja (bodajże Hopcroft'a) wyjaśnienia zjawiska nieadekwatności zmatematyzowanych teorii, przypominająca marksistowsko-leninowskie koło dziejów: oto na obecnym etapie matematyka wyczerpała swój potencjał i czeka na nową jakość w informatyce, która znowu wyzwoli jej możliwości. Zakłada się tu, że teorie informatyczne spełniły swoją rolę, owocując we wspaniałą praktykę. Ten model jest niestety fałszywy. Większość teorii informatycznych okazała się ortogonalna w stosunku do rozwoju informatycznych technologii: było teoriami post-factum, lub na zawsze pozostało nieadekwatne w stosunku do rzeczywistych problemów. Można mówić o sukcesach takich teorii jak gramatyki bezkontekstowe i zbiory/logiki rozmyte (ang. *fuzzy*[4]), ale nawet one mają drugorzędne znaczenie w pracy konstruktorów oprogramowania. Waite i Goos w swojej książce „Compiler construction" (Springer 1984) podają, że 1/3 prac na temat kompilatorów dotyczy analizy składniowej, podczas gdy w klasycznych kompilatorach

[2] Krążyła niegdyś anegdota o doktorancie, który skrócił czas działania algorytmu (powiedzmy) dziesięciokrotnie i o nazbyt pewnym siebie profesorze, który zarzucił mu, że jego osiągnięcie jest niewiele warte, bo złożoność obliczeniowa pozostała ta sama. Profesorowi zapewne nie przyszło do głowy, że to może złożoność obliczeniowa jest niewiele warta, skoro nie potrafi uwzględnić tak istotnych różnic. Wiele algorytmów, takich jak sortowanie, *hash coding*, operacje na *B*-drzewach, *locking and recovery*, etc. jest na tyle uwikłana w szczegóły techniczne, że formalne ich traktowanie jest praktycznie niemożliwe. Teoria złożoności daje niekiedy szansę odrzucenia a priori jakiegoś pomysłu (o ile złożoność jest np. wykładnicza), ale generalnie nie jest dla inżyniera efektywnym filtrem.

[3] Patrz np. materiały pierwszego ogólnopolskiego sympozjum „Naukowe Problemy Maszyn Matematycznych", Zakopane 1968, PWN 1970.

[4] Jest rzeczą ciekawą, że sukces *fuzzy logic* jest głoszony przez Japończyków; w USA, gdzie ta koncepcja się narodziła, stała się ona faktem z przeszłości bez istotnych śladów w rzeczywistości technologicznej. Można zatem podejrzewać, że *fuzzy logic* oznacza dzisiaj bardziej reklamowe hasło niż autentyczny transfer wiedzy naukowej do przemysłu.

parser zajmuje ok. 9–11% całości. W językach programowania baz danych, takich jak DBPL, ten udział spada poniżej 3%; jest to całkowity margines. Możliwość zastosowania matematyki wyzwoliła lawinę publikacji, natomiast problemy znacznie bardziej istotne, np. generowanie kodu, nie doczekały się odpowiedniej uwagi z powodu braku tej możliwości. „Wielkie" teorie, takie jak gramatyki dwupoziomowe, semantyka operacyjna (pierwsza metoda wiedeńska), semantyka denotacyjna (VDM), wielosortowe algebry do badania struktur danych, maszyny abstrakcyjne itp. okazały się w praktyce jałowe. Ogromne komercyjne sukcesy należą do produktów twórczości inżynierskiej, takich jak systemy operacyjne, systemy okienkowe, systemy graficzne, CAD, CAM i CASE, języki Pascal, C, C++, Smalltalk i inne języki obiektowe, 4GL, systemy zarządzania bazami danych, edytory, etc., gdzie udział zaawansowanych formalizmów w procesie wynalazczym był znikomy z prostego powodu: twórcy użytecznych systemów tych formalizmów nie znali (i dość często je lekceważyli), zaś ci, co je znali, prawie nigdy nie tworzyli rzeczy użytecznych.

Zwolennicy metod matematycznych, argumentując ich znaczenie w procesie twórczym, odwołują się do ich oddziaływania na intelektualne własności potencjalnego wynalazcy. Przykładem jest cytowana przez D. Gries'a (CACM 34/3, 1991, p. 45) opinia z książki Chandy i Misra: *...formalizm nie wypiera intuicji: on ją uzupełnia i wspiera...* W powszechnym odczuciu istnieje korelacja między zdolnościami i wiedzą matematyczną, a potencjałem twórczym; manifestacją tego odczucia jest zasada Ulama *matematyk zrobi to lepiej*[5]. Zdolności matematyczne są wrodzone i nie jest wykluczone, że są skorelowane ze zdolnościami do inżynierskiej wynalazczości, ale to nie oznacza przecież, że potencjał wynalazczy jest konsekwencją matematycznej wiedzy.

Stąd kontrowersje budzi kwestia, jaka ma być rola zmatematyzowanych teorii w procesie kształcenia informatyków. Programy nauczania są dość często przeładowane matematyką i zmatematyzowanymi informatycznymi teoriami. Jedną z przyczyn jest fakt, że wymiana użytecznej wiedzy w informatyce jest znacznie szybsza, niż naturalna wymiana kadr na wyższych uczelniach. Zwykłe programy nauczania są znacznie opóźnione w stosunku do stanu sztuki: stara kadra ma stare nawyki, stary system wartości naukowych, i po prostu, nie umie uczyć niczego innego. Wykładowcom – nie tylko w Polsce – nie chce się opanowywać nowej wiedzy, szczególnie, jeżeli wymaga ona żmudnego rozpoznania i testów na konkretnym systemie komputerowym; i bez tego są zmęczeni i przeciążeni. O ileż łatwiej jest wykładać klarowną teorię, powszechnie dostępną w podręcznikach.

Niekiedy motywacje są natury estetycznej. Dość często dydaktycy są sfrustrowani koniecznością przekazywania wiedzy ewidentnie arbitralnej, stanowiącej rezultat inżynierskiego chciejstwa i przypadkowych decyzji projektowych. Nie wiadomo, jak uczyć dziedzin owocujących w systemy-molochy, z ogromną liczbą wydawałoby się zbędnych lub słabo uporządkowanych możliwości, na dodatek dostępnych z języków przypominających wojskowe szyfry. Chciałoby się reguł, porządku, jednorodnej czystej koncepcji i generalnych praw. Motywem nauczania metod formalnych jest wiara, że zdyscyplinowane myślenie i sformalizowane postępowanie, które wymusza matematyka, daje podstawy dobrych inżynierskich zwyczajów. Powstaje jednak pytanie, czy wobec braku bezpośredniej użyteczności wiedzy matematycznej wystarczającą motywacją jest enigmatyczne oddziaływanie matematyki na psychologię i zwyczaje przyszłego inżyniera?

[5] Dodajmy, zasada szowinistyczna i nieprawdziwa; nie ma dowodów na to, że matematyk namaluje obraz lub napisze powieść.

Temat roli matematyki w procesie nauczania informatyki co jakiś czas odżywa na łamach Communications of the ACM (nie wykluczone, że za sprawą redaktora czasopisma, Petera Dennig'a). Dyskusja, którą sprowokowały artykuły E. Dijkstry (CACM 32/12, 1989, p. 1398) i D. Gries'a (CACM 34/3, 1991, p. 45), dwóch wybitnych teoretyków próbujących uzasadnić konieczność położenia większego nacisku na matematyczne wykształcenie studentów, świadczy o tym, że świat informatyki rzeczywistej i w tym zakresie przestaje teoretykom wierzyć. Czytelnik CACM ironizuje: Dijkstra, który napisał setki artykułów z dziedziny teorii programowania, nie używa komputera, nawet do pisania swoich artykułów. Niejeden praktyk w tym momencie zastanawia się, cóż on może wiedzieć o istocie tej dyscypliny, kto i jak mu tę istotę objawił, i gdzie leży granica między naukową twórczością a grafomanią? Dijkstra jest produktem swojej epoki, specjalistą od algorytmiczno-matematycznych łamigłówek. Są one ciekawe, ale mało istotne dla twórców współczesnych systemów komputerowych. Przy całym szacunku dla jego osiągnięć, trzeba naiwnych, którzy uwierzyliby w jego moralne prawo do misjonarskiego tonu w kwestii kształcenia współczesnego informatyka. Jakaż to różnica z D.E. Knuth'em, teoretycznym gigantem tworzącym tak praktyczne urządzenia jak TEX. W jego rozumieniu programowanie jest i pozostanie sztuką; sztuką nie teoretyczną, ale sztuką wykorzystania możliwości techniki i naturalnych ludzkich własności, a więc *sztuką kompromisów* niezbędnych do stworzenia użytecznego tworu.

Kompromis między stopniem zmatematyzowania teorii i jej praktyczną użytecznością może stać się również kluczem do dobrych teorii informatycznych. Wielu matematyków działających w informatyce chciałoby bronić poglądu, że teorie mogą być wyłącznie zmatematyzowane, ale w informatyce, przy obecnym stopniu złożoności tworów informatycznych jest to nierealne i często zbędne. Zadaniem teorii jest dostarczenie mechanizmów pozwalających budować abstrakcje odnośnie rzeczywistości oraz efektywnie wnioskować o stanie lub zjawiskach zachodzących w tej rzeczywistości. Sformalizowane teorie budowane w informatyce okazują się zbyt słabe w stosunku do tych celów. Nauki, takie jak medycyna, biologia, ekonomia wykształciły wiele niesformalizowanych teorii, które okazały się efektywne, a przez to użyteczne. Jestem przekonany, że informatyka również zmierza w kierunku budowania tego rodzaju teorii.

Przykładem takiej teorii w dziedzinie baz danych jest model-związek (ang. *entity-relationship model*), który uzyskał ogromny autorytet i znaczenie, głównie w dziedzinie projektowania zastosowań systemów baz danych. Problem, który stara się on rozwiązać, nie da się sformułować w terminach matematycznych. Oto informatyk, projektant systemu z bazą danych, ma zrozumieć jak jest zorganizowany i jak działa pewien wycinek rzeczywistości; niech to będzie zakład produkujący tworzywa sztuczne. Nie jest specjalistą z tego zakresu, prawie nic nie wie o tworzywach sztucznych. Ma do czynienia z tysiącami nowych pojęć, obiektów, wzorów chemicznych, technologii, organizacji etc. oraz zachodzących pomiędzy nimi różnorodnych związków. Jak ma myślowo opanować to monstrum? Model obiekt-związek jest zestawem prostych pojęć pozwalających budować abstrakcyjne wizje nie do końca zrozumiałego świata opisywanego przez dane, na różnych poziomach abstrakcji, wykrywać w nich anomalie, oraz wnioskować odnośnie transformacji tych wizji mających na celu usunięcie anomalii. Warstwa formalna tego modelu jest cienka i nieistotna[6], w istocie model odwołuje się do ludzkiej psychologii, wykorzystując naturalnie wbudowane w ludzkich mózgach mechanizmy percepcji i rozumienia

[6] Oczywiście, istnieje duża liczba jego pseudoformalizacji, powstających z braku lepszych pomysłów na „naukowe" publikacje.

świata. Ten nieformalny model stał się podstawą wielu systemów CASE. Teoretycy, tacy jak Stefano Ceri, potrafili go docenić, o czym świadczą znakomite publikacje. Do tego jest potrzebna zawodowa dojrzałość nie obciążona pędem do szybkiej naukowej kariery, chęć zrozumienia istoty problemu, i chęć szukania dróg jego efektywnego rozwiązania bez obciążenia się matematycznymi preferencjami.

Nie chciałbym uchodzić za wielkiego entuzjastę modelu obiekt-związek; można go jednak uważać za *efektywną informatyczną teorię*, która reprezentuje realistyczne podejście do problemów specyfikacji. Teoretycy zajmujący się „algebraiczną specyfikacją", „denotacyjną specyfikacją", specyfikacji za pomocą logik etc. wydają się ignorować fakt, że stan sztuki praktycznej w tym zakresie osiągnął duży poziom kompetencji i wyrafinowania. W metodologii projektowania baz danych wyłożonej w książce „Conceptual Database Design" (C. Batini, S. Ceri, S.B. Navathe, Benjamin/Cummings, 1992) specyfikacja powstaje jako rezultat analizy wymagań zamawiającego, zapisanych w postaci nieformalnego tekstu, z których sporządza się diagramy obiekt-związek (abstrakcje danych), diagramy funkcjonalne (abstrakcje programów) oraz inne informacje zapisane w bardziej jednoznacznej postaci językowej. Proces ustalenia nieformalnych wymagań i tworzenie diagramów powtarza się niekiedy wielokrotnie, aż do uzyskania zadowalającego rezultatu. Następnym etapem jest scalenie wielu diagramów uzyskanych od różnych grup użytkowników; ten proces może wymagać iteracji i nawrotów do poprzednich faz. W wyniku otrzymuje się wizję bazy danych i programów zastosowania, którą poddaje się nieformalnej, lecz zdyscyplinowanej analizie w celu wykrycia i usunięcia anomalii. Następnie tę wizję wzbogaca się w takie szczegóły, jak rodzaj i typ atrybutów, nazwa lub opis algorytmów przetwarzania etc. W wyniku powstaje wielowarstwowa specyfika zastosowania, zawierająca dane na różnym poziomie szczegółowości i różnym stopniu sformowania informacji. Ze względu na mocny związek z ludzką psychologią i percepcją, w tej metodologii nie ma miejsca na nietrywialną matematykę.

Wobec niemożliwości opanowania problemu przez matematykę, zamiast efektywnych informatycznych teorii powstają zmatematyzowane ideologie, próbujące sformułować i rozwiązać ten sam problem, ale w świecie urojonym, istniejącym wyłącznie w wyobraźni pomysłodawców lub kontynuatorów ideologii. Niekompetencja w zakresie informatyki praktycznej i różnorodne uproszczenia sprawiają, że z reguły są one ideologiami utopijnymi. Niestety ten fakt staje się ewidentny dopiero wtedy, gdy ideologia zostanie zrealizowana (notabene, do tego zwykle nie dochodzi). Wiemy doskonale, ile szkód wyrządziła realizacja utopijnej ideologii społecznej i nie ma powodu, aby w informatyce było inaczej. Przykładem jest chociażby model relacyjny baz danych, jeżeli tylko ktoś chciałby porównać założenia teoretyczne tego modelu z tym, co naprawdę zostało zrobione w systemach takich, jak: Ingres, Oracle, DB2 lub dBase. Zrobił to niejaki H.J. Baker, dochodząc do wniosku (CACM 35/4, 1992, pp. 16, 18), że model relacyjny baz danych opóźnił rozwój tej dziedziny technologicznej o 10 lat![7] Podobną zmatematyzowaną utopią okazują się dedukcyjne bazy danych (oparte na programowaniu w logice, a raczej na jego uproszczonej wersji zwanej Datalog), ale społeczność baz danych jest już nieufna. Tezy J.D. Ullman'a o wyższości dedukcji nad obiektowością spotkały się z powszechną krytyką, również ze strony osób zajmujących się dedukcją, którzy skłaniają się raczej ku konwergencji (patrz prace S. Abiteboul'a, R. Manthey'a, A.A.A. Fernandes'a i innych)[8]. Znacznie bardziej radykalne stanowisko można znaleźć w Laguna Beach Report[9] dotyczącym problemów z zakresu baz danych, którymi warto zajmować się w przyszłości. W raporcie tym 16 czołowych amerykańskich i niemieckich ekspertów ustaliło listę 25 tematów, porządkując ją od najważniejszych do najmniej istotnych. Generalna rekursja (czyli dedukcyjne bazy danych) zajmuje na tej liście ostatnie miejsce. Podobny eksperyment powtórzono w kwietniu 1993 w Wiedniu[10], i tym razem generalna rekursja zajęła ostatnie miejsce. Mimo ogromnych kontrowersji, jakie wzbudziły powyższe raporty (i zrozumiałego oburzenia środowisk akademickich) nie ma wątpliwości, że zaowocują one w konkretne decyzje dotyczące finansowania nauki i polityki naukowej.

Jaki może być powód tej niechęci świata komercyjnego do logiki matematycznej, której rola w informatyce – podobno – ostatnio wzrosła? Temat zastosowań logiki matematycznej w bazach danych, na które powołuje się bardzo wiele osób pragnących uzasadnić swoje teoretyczne poczynania (również Autorzy książki o logice algorytmicznej), poznałem dość dobrze, prowadząc przez ostatnie lata wykłady m.in. z tego zakresu na Uniwersytecie w Hamburgu. Nic istotnego z tej logiki nie ostało się w rzeczywistych systemach baz danych, takich jak Ingres i Oracle. Napisałem pod tymi systemami poważne aplikacje i spory kawałek oprogramowania, zwanego *gateway*, umożliwiający „przezroczysty" dostęp z języka DBPL (rozszerzona Modula-2) do systemów opartych na SQL; prowadziłem też cykl wykładów i ćwiczeń. Znam więc sprawę z obydwu stron (co generalnie jest rzadkością). Brak zastosowań logiki w tych systemach jest po prostu faktem nie podlegającym dyskusji. Powodem entuzjastycznych opinii o zastosowaniu logiki do baz danych jest wyłącznie to, że jest ona wehikułem do naukowych karier dla bardzo wielu ludzi, którym zwykle nie zależy na tworzeniu użytecznej wiedzy, a jedynie na owych karierach i niekiedy na frajdzie, jaką daje rozwiązywanie matematycznych łamigłówek. Dodam, że systemy takie jak NAIL! i LDL – w porównaniu z potrzebami występującymi w rzeczywistych systemach baz danych – pozostają, jak dotąd, tylko przedłużeniem teoretycznych zabaw. To samo dotyczy logik niemonotonicznych, opartych na kontrowersyjnych założeniach w rodzaju *closed world assumption*. Są to ciekawe obiekty z matematycznego punktu widzenia, ale tylko osoby nieświadome stanu informatyki praktycznej mogą twierdzić, że mogą one mieć jakiekolwiek znaczenie dla przyszłych systemów informatycznych. Nie będę śmiałym prorokiem, jeżeli powiem, że za parę lat cały ten nurt będzie się określać tak, jak dzisiaj określa się rezultaty projektu piątej generacji komputerów[11].

[7] Gwałtowny odpór temu twierdzeniu dała grupa obecnych bazo-danowych autorytetów (CACM 35/8, 1992, pp. 13–15), w większości tych, ...którzy swoje kariery zawdzięczają modelowi relacyjnemu. Powszechnie uważa się, że obecny zwrot w kierunku obiektowych baz danych jest kontynuacją tradycyjnych sieciowych baz danych, model relacyjny był naruszeniem tej ciągłości, a dzisiaj coraz bardziej staje się faktem z przeszłości. Mechanizmy modelowania pojęciowego są w modelu relacyjnym trudne do zaakceptowania, programowanie w SQL, moim zdaniem, jest równie paskudne jak programowanie w Codasyl-owskim DML. Dlaczego więc Baker nie miałby mieć racji?

[8] Można z góry założyć, że owa „konwergencja" będzie przysłowiową zupą na gwoździu: wobec inherentnych wad programowania deklaracyjnego (problemy z aktualizacją, z programowaniem interakcyjnych scenariuszy, z wydajnością itp.) z logiki zostanie tam tylko konwencja syntaktyczna.

[9] M. Stonebraker, E. Neuhold: The Laguna Beach Report. International Institute of Computer Science, Technical Report 1, Berkeley, Ca., 1989

[10] M. Stonebraker et al.: DBMS Research at a Crossroads: the Vienna Update. Proc. of 19th VLDB Conf. Dublin, 1993.

[11] W powszechnej opinii, jego rezultaty są bardzo mizerne. Przypomnę, że w projekcie tym kluczową rolę odegrały dwie zmatematyzowane ideologie: programowanie w logice i model relacyjny. Wydaje się jednak, że to doświadczenie nie dociera do niektórych osób ze środowisk akademickich.

Podstawowe niebezpieczeństwo, które niosą za sobą utopijne ideologie w dziedzinie informatyki, polega na tym, że są one intelektualnie atrakcyjne, stwarzają iluzję rozwoju oraz jasno określają doraźne cele naukowe. To powoduje, że angażują one najzdolniejsze, najbardziej aktywne osoby, głównie ze środowisk akademickich. Tak powstaje bezwładna lawina zmatematyzowanych prac naukowych, z których jedna napędza następną. Dzięki tym pracom ludzie realizują swoje ambicje zawodowe, robią kariery, stają się autorytetami. Motywacje wyznacza kolejna publikacja do *mojego osobistego* dorobku naukowego, wspomagająca moją osobistą karierę. Przez to bardziej istotna jest obecność aparatu matematycznego i asocjacje ze znanymi matematycznymi teoriami i autorytetami, niż jakikolwiek sens praktyczny. Każdy, dokładając swój kamyk do tej lawiny, nie myśli o jej globalnym kierunku; powstaje on jako wypadkowa doraźnych ludzkich interesów i przypadkowych pomysłów na nową definicję, nowy aksjomat, nowe twierdzenie. Byłoby nawet dziwne, gdyby ten wypadkowy kierunek prowadził do czegoś pożytecznego. Próby dopasowania utopijnej ideologii do realiów generują moc nowych problemów. Po jakimś czasie lawina zajmuje się już wyłącznie tymi wewnętrznymi problemami (aktualizacją, aksjomatyzacją, negacją, stratyfikacją, etc.); praktyka i rzeczywistość nic ją nie obchodzą. Lawina po latach musi opaść. Pozostaje po niej rumowisko coraz bardziej pokrywającej się kurzem bezwartościowej „klasyki", pozostają wątpliwej jakości „autorytety". Utopijna zmatematyzowana ideologia w dziedzinie informatyki rodzi ogromne marnotrawstwo najbardziej wartościowego potencjału twórczego.

Jednym ze źródeł utopijnych zmatematyzowanych ideologii jest wyznawana w niektórych środowiskach akademickich zasada prymatu teorii nad praktyką; dokładnie odwrotna, niż w fizyce. Jest ona równie manifestowana; z grubsza sprowadza się do tezy, że naukowcy-teoretycy tworzą teorie, które następnie są wdrażane przez rzemieślników-praktyków. Naukę tworzą wyłącznie teoretycy przy użyciu zaawansowanej matematyki; reszta należy do „biura projektowego" i oczywiście, nie jest nauką. Zasada ta jest szkodliwym mitem, który nie znalazł potwierdzenia w 40-letniej historii informatyki, ale przyczynił się do utrącenia lub zaniechania realizacji wartościowych projektów naukowych, oraz – co jest nawet ważniejsze – do zniszczenia motywacji wielu ludzi do prowadzenia badań eksperymentalnych [12]. W dziedzinach związanych z programowaniem niepodważalne i trwałe osiągnięcia należą do zespołów, które podjęły wieloletni trud budowy, udoskonalenia, testowania i tworzenia kolejnych wersji eksperymentalnych systemów. Jest to droga żmudna, często frustrująca, wymagająca dobrego pomysłu, dostępu do odpowiedniej klasy sprzętu i oprogramowania, i stabilnych perspektyw sporego zespołu w dość długim horyzoncie. Te czynniki są trudne nie tylko w biednych krajach takich jak Polska. Stąd ucieczki w „teorie", które można uprawiać indywidualnie, a owoce zbierać już po miesiącach. Należy jednak wyraźnie zdać sobie sprawę, że teorie te są substytutem: konsekwencją zrozumiałego pędu do naukowych karier, ale przecież nie jedyną a nawet nie główną podstawą wiedzy.

Budowanie wiedzy teoretycznej przy braku kompetencji w zakresie faktycznego stanu przedmiotu badań, stawianie się ponad ten przedmiot, brak najdrobniejszych prób skonfrontowania tworzonych teorii z rzeczywistością w każdej innej dziedzinie nauk stosowanych byłby naruszeniem zasad obowiązujących w nauce. Chyba tylko informatyka jest tu wyjątkiem, i nie dlatego, że jest ona w jakiś szczególny sposób różna od tych nauk. Przyczyny są genealogiczne. Informatyka w sensie kadrowym wyrosła z matematyki i ten zmatematyzowany styl uprawiania informatyki jako nauki obrósł w autorytety i instytucje: cały samopodtrzymujący się i samoreprodukujący się świat. Świat ten nie wiedzieć kiedy zmienił się ze świata wielkich naukowych nadziei w świat pozorów wiedzy i urojonych wartości. W ten sposób matematyka – o ironio, królowa nauk! – przez swój autorytet, hermetyczność i powszechną akceptację braku bezpośredniego związku pomiędzy teorią i praktyką, stała się idealnym nośnikiem pseudonaukowej twórczości w dziedzinie informatyki. Egzotycznie brzmiące pojęcia, powołanie się na wybitne matematyczne teorie i autorytety, pozory dużej ogólności i uniwersalności, wzory, definicje, twierdzenia itp. są zbyt często zasłoną ukrywającą niefachowość, niekompetencję, karierowiczostwo, brak dobrego pomysłu, i brak doświadczeń z jakimkolwiek działającym systemem.

Pora na wnioski. Informatykę zalewa zmatematyzowana grafomania, produkowana pod hasłami „sztucznej inteligencji", „systemów ekspertowych", „dedukcyjnych baz danych", „programowania w języku logiki", „inżynierii oprogramowania", „weryfikacji programów", „współbieżnego przetwarzania" etc. etc. Jest to okoliczność, która bardzo zniechęca świat informatyki rzeczywistej do rezultatów uzyskiwanych na drodze formalnej. Jestem zdania, że świat informatyczny, a szczególnie osoby uważające się za teoretyków informatyki, powinni zdać sobie sprawę z przyczyn patologii i określić zasady przeciwdziałania upadkowi autorytetu matematyki, który jest szczególnie widoczny w dziedzinach programowania, baz danych i inżynierii oprogramowania. Podsumowując poprzednią dyskusję, moja lista patologii związanych z zastosowaniem matematyki w informatyce jest następująca:

- Karierowiczostwo i oportunizm: do niedawna najłatwiejsza droga do kariery naukowej w informatyce polegała na publikowaniu dużej liczby zmatematyzowanych prac. Ich użyteczność nie podlegała istotnym ocenom, często z powodu braku kompetencji ciał oceniających (np. rad naukowych), złożonych głównie z „teoretyków".

- Wypaczanie się informatycznej teorii w kierunku czystej matematyki: nadmierne generalizacje, nieuzasadnione specjalizacje, abstrahowanie od istotnych cech problemów, które teoria ma rozwiązać.

- Wypaczanie się sformalizowanej informatycznej teorii w kierunku utopijnej ideologii (casus logiki algorytmicznej dla programistów).

- Rozwiązywanie problemów zastępczych, często drugorzędnych wewnętrznych problemów jakiejś zmatematyzowanej ideologii, w sytuacji gdy zasadnicze problemy związane z tą ideologią pozostają nie rozwiązane (casus problemu negacji w programowaniu w logice).

- Doktrynalny stosunek do sformalizowanej ideologii, uniemożliwiający jej ewolucję: jest on spowodowany uzasadnioną obawą, że zmiana aksjomatów może zepsuć już osiągnięte rezultaty.

- Teorio-centryczny szowinizm, występujący wtedy, gdy grupa teoretyków uzyskuje dominację w pewnym środowisku lub na pewnym forum. Ludzie ci, podtrzymując się wzajemnie w swoich poglądach i niezauważający swojej niekompetencji, lansują zmatematyzowaną fikcję, niszcząc przy tym wartości autentyczne.

Z drugiej strony, niebezpieczeństwo tkwi również w nadmiernie pragmatycznym stosunku do badań naukowych. Nauka nie

[12] Z teorio-centrycznym szowinizmem teoretyków informatyki z mojego bliższego i dalszego otoczenia zetknąłem się wielokrotnie, i dysponuję licznymi jego przykładami. Chyba najbardziej spektakularnym z nich było utrącenie (pośrednio) w 1980 r. przez ówczesną Radę Naukową IPI PAN projektu LINDA, który miał wiele sukcesów, ale zdaje się nie pasował do lansowanego wtedy modelu „podstaw informatyki".

może wyłącznie odpowiadać na zapotrzebowanie ze strony praktyki, gdyż w ten sposób można zniszczyć jej podstawową funkcję, jaką jest tworzenie nowych jakości intelektualnych. Drogi rozwoju intelektualnego są zagmatwane i często nie sposób ustalić wszelkie źródła postępu. Z tego względu nie zawsze badania naukowe muszą mieć bezpośrednie efekty praktyczne. W wielu przypadkach wystarczającym motywem napędzającym naukę jest to, że ktoś usilnie pragnie uzyskać odpowiedź na pytanie *jak to jest?* bez zadawania sobie pytania *do czego to się przyda?* Tak przecież powstała filozofia i wiele innych nauk. Pomiędzy działalnością inżynierską mającą na względzie cele komercyjne i „czystą teorią" istnieje ogromne pole dla działalności naukowej. Mieszczą się w nim systemy budowane dla celów eksperymentalnych, którym przyświecają motywacje: *jak to zrobić?, czy uda się to zrobić? czy ludzie to zaakceptują?* itd. Jest to droga sprawdzona i powszechnie stosowana na całym świecie. Ważnym efektem ubocznym eksperymentów w konkretnym środowisku komputerowym jest wytworzenie kadry, która może stanowić doradczo-weryfikacyjną bazę dla sfery polityki, planowania i produkcji informatycznej. Jestem zdania, że tylko nieliczni teoretycy informatyki są w stanie dobrze pełnić tę funkcję. Eksperymenty często owocują w wiedzę syntetyczną; mogą być też podstawą budowy efektywnych zmatematyzowanych teorii. Jestem jednak przekonany, że droga odwrotna – od zmatematyzowanych teorii do praktyki – jest na obecnym etapie rozwoju informatyki drogą prowadzącą do nikąd.

II Konferencja EDI w Polsce

W czerwcu br. odbyła się w Arturówku pod Łodzią I Krajowa Konferencja EDI (*Electronic Data Interchange* – Elektroniczna Wymiana Danych). EDI polega na przesyłaniu komunikatów elektronicznych, stanowiących zdematerializowane odpowiedniki tradycyjnych dokumentów papierowych, pomiędzy systemami komputerowymi partnerów gospodarczych przy wykorzystaniu sieci telekomunikacyjnych. Coraz powszechniejsze stosowanie EDI na Zachodzie oraz ogromne korzyści osiągane przez użytkowników powodują rosnące zainteresowanie tą klasą systemów informatycznych również w Polsce.

Jeden z końcowych wniosków I Konferencji, adresowany do organizatorów, dotyczył potrzeby przygotowania kolejnej jej edycji. Realizując ten postulat, podjęto już prace nad organizacją II Konferencji EDI w Polsce. Odbędzie się ona, podobnie jak poprzednia, również w ośrodku zlokalizowanym w otoczeniu pięknego lasu **w Arturówku pod Łodzią** (dojazd autobusem miejskim z dworca Łódź-Fabryczna)

w dniach od 30 maja do 1 czerwca 1994 r.

Przedmiotem obrad będą problemy budowania, wdrażania i eksploatacji systemów EDI oraz przegląd realnych dokonań i doświadczeń w tym zakresie w naszym kraju.

W programie II Konferencji są przewidywane, poza prezentacją referatów i dyskusjami panelowymi, również spotkania grup użytkowników w sekcjach branżowych (m.in. przemysł, bankowość, transport, administracja) oraz wystawa oprogramowania używanego w EDI. Organizatorzy przewidują także liczniejszy, niż w bieżącym roku, udział gości zagranicznych, którzy podzielą się własnymi doświadczeniami z zakresu EDI.

Konferencja jest adresowana do wszystkich obecnych i przyszłych użytkowników systemów EDI, zwłaszcza reprezentujących przedsiębiorstwa przemysłowe, handlowe, bankowe i transportowe zaangażowane w operacje zagraniczne.

Szczegółowych informacji w sprawie konferencji udziela:

prof. dr hab. Marian Niedźwiedziński
ul. Szklana 2/14 m. 3, 91-844 Łódź
tel., faks: (0-42) 55-67-57

A case study: programming in Object Windows

dokończenie ze s. 4

- Working with the documentation and studying how-to literature with the accompanying extraction of relevant items that should be memorized (about 2 hours). Here, only basic concepts and terminology should sparingly be marked for memorization with SuperMemo.
- Programming training, perhaps including the development of the code for the ultimate project, with extraction of items useful in programming tasks and in overcoming most typical programming hurdles (about 5 hours). Here, only the most frequently used details of the language and the development environment should be considered. Particular emphasis should be put on specifying items that solve bottleneck problems that have been encountered in the course of the practice.
- Development of a SuperMemo database and repetitions of the outstanding items (about 1 hour).

Obviously, the time needed for studying the documentation and the time needed for SuperMemo repetitions can decline rapidly with the advancement of the project, especially in programmers with a good background on the considered language and environment. Similarly, the availability of customized, ready-made SuperMemo databases could cut the third component of the training by up to 70% time.

The figures presented earlier in reference to the effectiveness of using SuperMemo in software development training may sound encouraging; however, it is not easy to convince an effective programmer with a firmly established modus operandi to adopt new solution, especially if they touch the neglected and under-appreciated field of training.

The major obstacle in popularization of SuperMemo among programmers seems to be the fact that development of well-structured databases that could result in a sharp increase in productivity is not as easy a task as it might seem at first. Therefore, an inexperienced user of SuperMemo software may quickly become disillusioned with rapid accumulation of repetitions resulting from wrong selection of the material, as well as from formulating items that do not comply with the principles of effective learning as stated in the definition of the SuperMemo method. However, the author is absolutely positive that it is only the question of time before SuperMemo becomes a standard tool used in routine training of a modern programmer.

REFERENCES

[1] Bell D., Morrey I., Pugh J.: Software engineering. Prentice Hall, 1987
[2] Borland Int.: Object Windows. Programming Guide. Borland International, 1992
[3] Petzold Ch.: Programming Windows 3.1. Microsoft Press, 1992
[4] Woźniak P.A.: Optimization of learning. A new approach and computer application, 1990
[5] Woźniak P.A., Biedalak K.: Metoda SuperMemo. Optymalizacja procesu uczenia się. *Informatyka* 1992, nr 10, s. 1
[6] Woźniak P.A., Gorzelańczyk E.J.: Optimal scheduling of repetitions in paired-associate learning. Acta Neurobiologiae Experimentalis. 1992, Vol. 52, p. 189,
[7] Woźniak P.A.: Speed-learning with SuperMemo for Windows. SuperMemo World. 1993
[8] Woźniak P.A.: A new method of fast learning and its software implementation. Polish Technical Review. 1993, Vol. 1, p. 8.

JANUSZ GWIAZDA
North Vancouver
Kanada

Ośrodek obliczeniowy firmy ubezpieczeniowej w Kanadzie

Ośrodek obliczeniowy, który chciałbym opisać, a zwłaszcza jego organizacja pracy, należy do niemodnego już typu opartego na dużym komputerze (IBM) o typowej gwiaździstej konfiguracji. Mimo to sądzę, że stosowana organizacja pracy i metody zapewniające niezawodne działanie systemów są na tyle nie związane ze sprzętem komputerowym i na tyle typowe dla Kanady, że warto je bliżej poznać. Nie sądzę również, aby problemy te były powszechnie znane w Polsce.

Nieco o samym zastosowaniu

Przedstawienie organizacji pracy ośrodka obliczeniowego rozpocznę od pobieżnej prezentacji samej instytucji dla której ośrodek ten pracuje. Uważam, że taka ilustracja ułatwi wyobrażenie o skali i wadze przetwarzania, które realizuje ośrodek określony w strukturze firmy skrótem ISD (*Information System Division*).

WCB – *Worker's Compesation Board* (w wolnym przekładzie Centrum Odszkodowań Pracowniczych) – instytucja podobna w działaniu do części polskiego ZUS, ma w swojej strukturze wspomniany ośrodek ISD. WCB jest firmą semi-rządową: kierownictwo (prezydenci i rada nadzorcza) jest powoływane przez ministra pracy, członkami rady są reprezentanci pracodawców, związków zawodowych oraz rzecznicy interesu publicznego. Przez „semi" chciałem podkreślić brak zależności od budżetu prowincji (lub federalnego) Kanady. WCB prowadzi gospodarkę finansową opartą na składkach pracodawców, a że są to składki obowiązkowe, ustanowione przez prawo (notabene wszystkie prowincje Kanady mają podobne akty prawne różniące się co najwyżej szczegółami technicznymi, więc dopóki istnieją pracodawcy to WCB ma z czego wypłacać odszkodowania i prawidłowo funkcjonować. Stale jednak istnieje konieczność planowania wpływów i wydatków, a w szczególności poprawnego przewidywania ich przyszłych wielkości. Wysokość składek jest ustalana na przyszłe okresy i zależy od statystyk wypadków w danym typie przedsiębiorstw, a również w konkretnym przedsiębiorstwie (jeśli jest odpowiednio duże, to samo stanowi grupę). Dla przykładu, w przemyśle drzewnym i górnictwie składki są najwyższe, ale różnią się tam w poszczególnych przedsiębiorstwach. Podstawowa stawka mnożona przez roczny fundusz płac daje wysokość rocznej kwoty, jaka powinna być wpłacona do WCB jako składka ubezpieczeniowa pracowników. Na marginesie warto zwrócić uwagę, że nie chodzi tu o tzw. indywidualne ubezpieczenia od wypadków, które są często również organizowane przez pracodawców (zbiorowe ubezpieczenie jest wielokrotnie tańsze niż indywidualne). Prawo nie tyle zmusza pracodawcę do płacenia składek, ile chroni pracownika – ubezpieczenie pracownika jest automatyczne, niezależnie czy pracodawca płaci składki czy nie. WCB w razie wypadku przy pracy zapewnia pracownikowi rekompensatę i jeśli pracodawca nie płacił składek przed wypadkiem, to musi pokryć pełne koszty związane z odszkodowaniem. Odszkodowanie takie czasem sięga setek tysięcy dolarów, a WCB ma moc prawną do wszczynania postępowania licytacyjnego włącznie. Inaczej mówiąc, od egzekucji nie da się wykręcić i dlatego lepiej płacić składki ubezpieczenia pracowników, ponieważ to znacznie zmniejsza koszty działania przedsiębiorstwa.

Odszkodowanie jest dwojakiego rodzaju: albo jednorazowe jako zwrot utraconych zarobków (w wyniku niezdolności do pracy), albo jako renta dożywotnia za utratę zdrowia (mierzona procentami jego utraty). Osobną kategorię stanowią renty dla wdowy czy dzieci w wyniku śmiertelnego wypadku – renta taka jest wypłacana dla dzieci do uzyskania pełnoletności (18 lub 25 lat, jeśli jest kontynuowana nauka), zaś dla wdowy dożywotnio lub do jej ponownego zamążpójścia (wówczas renta ustaje po upływie dwóch lat od daty ślubu).

WCB istnieje już 75 lat, najstarsza i jeszcze aktywna renta sięga czasów I wojny światowej, zbiory archiwalne sięgają dziesiątków milionów dokumentów, np. rocznie rozpatrywanych jest ok. 100 tys. nowych *claims* – roszczeń za wypadek przy pracy. Ich większość to jednorazowe niewielkie sumy za czasową niezdolność do pracy. Oznacza to, że wypłacane co miesiąc renty (albo jak to się nazywa – *pension*), są wypłacane ok. 30 tys. osobom, a niektóre z wypłat są z kilku tytułów, np. osoba uległa ponownemu wypadkowi i otrzymała nową *pension*, albo poprzedni wypadek w wyniku rozwoju choroby powoduje nową utratę zdrowia (np. uraz głowy bezpośrednio wywołał utratę słuchu, a po jakimś czasie przyczynił się do utraty węchu).

Rozproszenie przetwarzania

WCB jest jedyną instytucją, w której można (i należy) ubezpieczyć pracowników od wypadków przy pracy (na marginesie ciekawostka – takiego ubezpieczenia nie można dokonać dla pracowników związanych z przemysłem rozrywkowym). Jest ona jedyną w prowincji – a prowincja jest obszarowo wielokrotnie większa od Polski. Wspominam o tym, gdyż jest to istotne ze względu na zasady działania obsługi rentowej. Po pierwsze, na terenie prowincji rozmieszczone są biura, gdzie składa się roszczenie albo załatwia związane z nim sprawy, ale nie ma obowiązku rejonizacji (co byłoby nonsensem przy częstej zmianie adresu przez Kanadyjczyków), tzn. w każdym biurze można załatwiać swoje roszczenia. Brak rejonizacji wymusza możliwe dwojakie rozwiązanie:

- albo dokumenty są gromadzone w jednym miejscu, a dostęp do nich jest z wielu oddalonych miejsc,
- albo gromadzi się je tam, gdzie powstały (gdzie pierwszy raz roszczenie zostało zgłoszone), a dostęp do nich jest z dowolnego innego miejsca. Jak można łatwo się domyśleć, wybrano rozwiązanie pierwsze jeszcze wiele lat temu, kiedy było ono niewątpliwie tańsze. Widzimy więc typowy gwiazdzisty system przetwarzania z jednym centralnym biurem gromadzącym

Spis treści rocznika INFORMATYKA 1993

Artykuły problemowe

	nr	s.
ADAMSKI WŁODZIMIERZ: Sieci komputerowe w polskim przemyśle lotniczym	10	1
BARTOSZEK DANUTA: ECHO – komputerowy serwis informacyjny	5	7
BENEDYKT ZBIGNIEW: Co daje obiektowe podejście do analizy i projektowania systemów informatycznych?	11	10
BENDIG-WIELOWIEJSKI ANDRZEJ, RUCIŃSKI ANDRZEJ, SARAD JERZY: Zastosowania wielofunkcyjnych elektronicznych kart bezstykowych	2	13
BERENT JAROSŁAW A.: TIFF – standard grafiki pikselowej	2	16
BERUS TOMASZ, MACUKOW BOHDAN: Od sieci neuronowych do neurokomputerów	1	1
BIJAŁD MAGDALENA, GWÓŹDŹ MARCIN, ZIELIŃSKI KRZYSZTOF: Rozproszone przetwarzanie w systemach otwartych. Model ANSA	3	17
BOCHIŃSKI JAROSŁAW: Matematyka w tworzeniu oprogramowania	2	18
CABAN DARIUSZ: Środki łączności w przemysłowych rozproszonych systemach sterowania	1	19
CZAJKOWSKI GRZEGORZ, ZIELIŃSKI KRZYSZTOF: STAND88 – język współbieżnego programowania w logice	1	13
CZAJKOWSKI GRZEGORZ, ZIELIŃSKI KRZYSZTOF: Linda – środowisko do przetwarzania równoległego i rozproszonego w sieciach stacji roboczych	5	1
DĄBROWSKA IWONA, DEPOWSKA DOROTA: Podstawowe problemy rozpoznawania pisma drukowanego w systemach off-line	2	22
DEPOWSKA DOROTA, DĄBROWSKA IWONA: Podstawowe problemy rozpoznawania pisma drukowanego w systemach off-line	2	22
DRĄŻEK ZYGMUNT, REUSCH PETER J.A.: System FCS jako środowisko programowe do budowy systemów wspomagania decyzji	3	23
FUGLEWICZ PIOTR: Języki czwartej generacji a systemy CASE	2	11
FUGLEWICZ PIOTR, OSTASZEWSKA ANNA: PROGRESS-em przez ocean danych	3	1
GAIK DARIUSZ, PRZYBYLSKI ANDRZEJ: Rekomendacje narzędzi graficznych dla środowiska APW	5	17
GŁAZEK WOJCIECH: Założenia, projekt i realizacja sieci transmisji danych KOLPAK	11	24
GOLIŃSKI JAN: Informatyka gospodarcza	9	10
GOLIŃSKI JAN: Problemy projektowania systemów informacyjnych	5	19
GOLIŃSKI MICHAŁ: Rynki elektroniczne – podstawowe pojęcia, obszary zastosowań i aspekty ekonomiczne	7	16
GRZELAK TOMASZ: Systemy wspomagania projektowania w przemyśle reklamowym	7	22
GWIAZDA JANUSZ: Ośrodek obliczeniowy firmy ubezpieczeniowej w Kanadzie	12	12
GWÓŹDŹ MARCIN, BIJAŁD MAGDAENA, ZIELIŃSKI KRZYSZTOF: Rozproszone przetwarzanie w systemach otwartych. Model ANSA	3	17
JAWORSKI WOJCIECH M., KURAŚ MARIAN, ZALIWSKI ANDRZEJ: infoMapy – inna metoda gromadzenia wiedzy	11	1
KALINOWSKI TOMASZ: Programowanie systemów transputerowych	4	1
KASPRZYK ARTUR: Obiektowo zorientowane analiza i projektowanie		
Cz. 1	9	1
Cz. 2	10	21
KĘDZIERSKI STANISŁAW: Modelowanie obiektów dynamicznych	12	16
KISIELNICKI JERZY: Informacyjna infrastruktura zarządzania – potrzeby i tendencje zmian	2	8
KOMOROWSKI WITOLD, GODULA KRZYSZTOF: Analiza stosowania rozkazów mikroprocesora 8086	11	21

	nr	s.
KOSMOWSKA-MISZALSKA DANUTA, TROJANOWSKI PIOTR: INFORMIX – fikcje i fakty	8	5
KORBEL LUCYNA, MIŁOSZ ELŻBIETA, MIŁOSZ MAREK: Systemy zarządzania, relacyjnymi bazami danych oraz ich przydatność do budowy informatycznych systemów zarządzania	4	8
KOTULA JACEK, ZYGUŁA JAROSŁAW: Model konwersacyjny CAL w środowisku KnowledgePro	4	21
KOTZIAN LESZEK: Prosta integracja środowiska sieci NetWare ze światem UNIX	5	14
KURAŚ MARIAN, JAWORSKI WOJCIECH M., ZALIWSKI ANDRZEJ: infoMapy – inna metoda gromadzenia wiedzy	11	1
LAURENTOWSKI ALEKSANDER, ZIELIŃSKI KRZYSZTOF: SR – obiektowo zorientowany język programowania rozproszonego	6	15
MACUKOW BOHDAN, BERUS TOMASZ: Od sieci neuronowych do neurokomputerów	1	1
MATLAK ANDRZEJ: Prawne środki ochrony programów komputerowych, możliwe kierunki zmian w USA	4	25
MIŁOSZ ELŻBIETA, KORBEL LUCYNA, MIŁOSZ MAREK: Systemy zarządzania relacyjnymi bazami danych oraz ich przydatność do budowy informatycznych systemów zarządzania	4	8
MIŁOSZ MAREK: Mikrokomputerowa symulacja systemów – GPSS/PC	3	11
MIŁOSZ MAREK, KORBEL LUCYNA, MIŁOSZ ELŻBIETA: Systemy zarządzania relacyjnymi bazami danych oraz ich przydatność do budowania informatycznych systemów zarządzania	4	8
MIŁOSZ MAREK: Quo vadis?, czyli ciąg dalszy o wyborze relacyjnej bazy danych	12	20
MYSZKOWSKI KAROL, REJ PAWEŁ, WOJDALA ANDRZEJ: Wspomaganie sprzętowe syntezy realistycznych obrazów	8	21
NOWACKI ZBIGNIEW A.: Szablony klas i listy elastyczne w pakiecie narzędziowym POLY do języka C	10	6
OSTASZEWSKA ANNA, FUGLEWICZ PIOTR: PROGRESS-em przez ocean danych	3	1
PARDYAK PRZEMYSŁAW, ZIELIŃSKI KRZYSZTOF: Emerald – język i system rozproszonego programowania obiektowego	7	1
PLATA-PRZECHLEWSKI TOMASZ, WRYCZA STANISŁAW – Badania ankietowe Cz. 1. Ocena poziomu zastosowań informatyki dla celów zarządzania w Polsce	6	6
Cz. 2. Główne zagadnienia projektowania i użytkowania systemów informatycznych	8	16
PORWIK PIOTR: Multimedialne bazy wiedzy – rekomendacje	10	13
PROKOP JAROSŁAW: System CHARLIE II – modelowanie i animacja sylwetek ludzkich	9	24
PRZYBYLSKI ANDRZEJ, GAIK DARIUSZ: Rekomendacje narzędzi graficznych dla środowiska APW	5	17
REJ PAWEŁ, MYSZKOWSKI KAROL, WOJDALA ANDRZEJ: Wspomaganie sprzętowe syntezy realistycznych obrazów	8	21
REUSCH PETER J.A., DRĄŻEK ZYGMUNT: System FCS jako środowisko programowe do budowy systemów wspomagania decyzji	3	23
RUCIŃSKI ANDRZEJ, BENDIG-WIELOWIEJSKI ANDRZEJ, SARAD JERZY: Zastosowania wielofunkcyjnych elektronicznych kart bezstykowych	2	13
RYŻKO JAN: Projekt nowych standardów. Zestaw znaków w przetwarzaniu i przesyłaniu informacji	11	14
SACHA KRZYSZTOF M.: Projektowanie oprogramowania systemów wbudowanych	7	8
SARAD JERZY, BENDIG-WIELOWIEJSKI ANDRZEJ, RUCIŃSKI ANDRZEJ: Zastosowania wielofunkcyjnych elektronicznych kart bezstykowych	2	13
SIENKIEWICZ PIOTR: Systemy ekspertowe w zarządzaniu	2	1
SROKA HENRYK: Systemy wspomagania decyzji z bazą wiedzy	4	17
SUBIETA KAZIMIERZ: Logika teoretycznych urojeń	12	5
SZYJEWSKI ZDZISŁAW: Klasyfikacja narzędzi wspomagania CASE	4	13
TOPOLEWSKI ZYGMUNT: Klucze szyfrujące w metodzie RSA	3	25
TROJANOWSKI PIOTR, KOSMOWSKA-MISZALSKA DANUTA: INFORMIX – fikcje i fakty	8	5
WEISS ZBIGNIEW: Rozproszone systemy operacyjne	6	1
WIECZERZYCKI WALDEMAR: Obiektowe bazy danych	1	8
WIECZERZYCKI WALDEMAR: Zarządzanie współbieżnością transakcji w obiektowych bazach danych	3	5
WIECZERZYCKI WALDEMAR: Wielowersyjne bazy danych	8	1
WIECZOREK MIROSŁAW: Podstawowe zagadnienia programowania zorientowanego obiek-		

	nr	s.
towo – polimorfizm i dziedziczenie klas w języku C++	10	16
WIERZBICKI MAREK: Tworzenie obiektów przez dziedziczenie mono- i polimorficzne	9	12
WITKOWSKI TADEUSZ: Algorytm rozpoznawania problemu szeregowania zadań	6	20
WITKOWSKI TADEUSZ: Komputerowe wspomaganie decyzji o zakupie produktu	8	9
WITKOWSKI TADEUSZ: Wspomaganie decyzji o wyborze zintegrowanego systemu CAD/CAM	9	16
WOJDALA ANDRZEJ, MYSZKOWSKI KAROL, REJ PAWEŁ: Wspomaganie sprzętowe syntezy realistycznych obrazów	8	21
WOŹNIAK PIOTR: SuperMemo as a new tool increasing the productivity of a programmer. A case study: programming in Object Windows	12	1
WRYCZA STANISŁAW, PLATA-PRZECHLEWSKI TOMASZ: Badania ankietowe		
Cz. 1. Ocena poziomu zastosowań informatyki dla celów zarządzania w Polsce	6	6
Cz. 2. Główne zagadnienia projektowania i użytkowania systemów informatycznych	8	16
ZABRODZKI JAN: Grafika komputerowa w programie studiów	7	26
ZALIWSKI ANDRZEJ, KURAŚ MARIAN, JAWORSKI WOJCIECH M.: infoMapy – inna metoda gromadzenia wiedzy	11	1
ZIELIŃSKI KRZYSZTOF, CZAJKOWSKI GRZEGORZ: STAND88 – język współbieżnego programowania w logice	1	13
ZIELIŃSKI KRZYSZTOF, BIJAŁD MAGDALENA, GWÓŹDŹ MARCIN: Rozproszone przetwarzanie w systemach otwartych. Model ANSA	3	17
ZIELIŃSKI KRZYSZTOF, CZAJKOWSKI GRZEGORZ: Linda – środowisko do przetwarzania równoległego i rozproszonego w sieciach stacji roboczych	5	1
ZIELIŃSKI KRZYSZTOF, LAURENTOWSKI ALEKSANDER: SR – obiektowo zorientowany język programowania rozproszonego	6	15
ZIELIŃSKI KRZYSZTOF, PARDYAK PRZEMYSŁAW: Emerald – język i system rozproszonego programowania obiektowego	7	1
ZYGUŁA JAROSŁAW, KOTULA JACEK: Model konwersacyjny CAL w środowisku KnowledgePro	4	21

Z kraju

	nr	s.
The Third International Conference Information Systems Developers Workbench. Sopot 22–24 września 1992 r., – *Stanisław Wrycza, Witold Rybiński*	5	22
Założenia wdrażania w RP Systemu Rozwoju Zawodowego na licencji Brytyjskiego Towarzystwa Komputerowego – oprac. *Piotr Fuglewicz, Paweł Gizbert-Studnicki*	5	25
IV Walny Zjazd PTI	8	27
Fenomen OPTIMUSA	12	22
Konrad Zuse profesorem honorowym Uniwersytetu Szczecińskiego – *W.K.*	12	23

Ze świata

Rynek mikroprocesorów X86 – *J.R.*	3	27
Szybki rozwój systemów Windows – *J.R.*	5	27

Konferencje

CeBIT'93 – *WK*	1	12
Kongres IFIP'93	5	21
I Krajowa Konferencja EDI	6	28
Grafika komputerowa – Przetwarzanie obrazów GKPO'94	8	26
II Konferencja Polskiego Klubu Użytkowników PROGRESS-a	8	28
SYSTEMS'93	10	26
Oprogramowanie Komputerowych Systemów Czasu Rzeczywistego	10	27
Drukarka SJ-144	11	13
The Fourth International Conference Information Systems Development – ISD'94	12	s. IV wew. nr.
II Konferencja EDI w Polsce	12	11

Recenzje

Inteligentne bazy danych – *Stefan Januszewski, Maria Januszewska*	1	27
Algorytmika – sedno informatyki – *Zbigniew Weiss*	2	28

Nowe książki

Wydawnictwa Naukowo-Techniczne	4	7
Nowy kwartalnik informatyczny „Machine Graphics & Vision"	6	5
Wydawnictwa Naukowo-Techniczne	6	27
Wydawnictwa Naukowo-Techniczne	8	26
Akademicka Oficyna Wydawnicza RM	9	27
Akademicka Oficyna Wydawnicza RM	10	25
Wydawnictwa Naukowo-Techniczne	10	20
Wydawnictwa Naukowo-Techniczne	11	27

Wskazówki dla Autorów

Nadsyłane artykuły nie mogą być publikowane lub przeznaczone do opublikowania w innych czasopismach.

Materiał oprócz tekstu zasadniczego powinien zawierać:

- ★ *krótki życiorys zawodowy Autora i jego zdjęcie,*
- ★ *wykaz literatury,*
- ★ *tabele,*
- ★ *materiał ilustracyjny (rysunki, zdjęcia czarno-białe, wydruki) dołączony do artykułu (nie wklejać materiału w tekst),*
- ★ *podpisy pod ilustracje.*

Na osobnej stronie prosimy podać tytuł naukowy, imię i nazwisko, nazwę zakładu pracy, adres domowy (koniecznie!), numer telefonu oraz informację, jaką drogą przekazać honorarium – kasa Wydawnictwa, poczta, bank.

Tekst artykułu prosimy dostarczyć w formie maszynopisu lub wydruku komputerowego (pisany jednostronnie, z podwójnym odstępem – bardzo ważne! – czyli 30 wierszy na stronie i 60 znaków w wierszu).

Wykaz literatury w porządku alfabetycznym.

Tabele – każda na osobnej stronie – powinny być ponumerowane, opatrzone tytułem oraz ściśle związane z tekstem (zaznaczone miejsce tabeli w tekście).

Rysunki – winny być czytelne (zaznaczyć ich miejsce w tekście), mogą być wykonane ołówkiem.

Zdjęcia – czarno-białe, kontrastowe, na błyszczącym papierze.

Wydruki – czytelne, kontrastowe, wykonane na białym papierze. Format – maksymalnie 18 cm w podstawie.

Każde pierwsze słowo opisu rysunków powinno być pisane dużą literą.

Podpisy pod ilustracje powinny być napisane na oddzielnej stronie, oprócz kolejnego numeru powinny zawierać tytuł (rysunku, zdjęcia, wydruku) i ew. legendę dotyczącą poszczególnych elementów.

Autor opublikowanego w INFORMATYCE artykułu otrzymuje honorarium i bezpłatny egzemplarz okazowy.

Materiałów nie zamówionych redakcja nie zwraca.

meditronik SPÓŁKA z o.o.

UMC UNITED MICROELECTRONICS CORPORATION

HEWLETT PACKARD COMPONENTS

BOURNS

- CZĘŚCI ELEKTRONICZNE
- KOMPUTERY PS/1,PS/2
- DRUKARKI HP
- INSTALACJE SIECI KOMPUTEROWYCH

Partnerzy handlowi: ANALOG DEVICES, ITT, MOTOROLA, SAMSUNG, TELEFUNKEN i inni

- UKŁADY PAMIĘCI
- UKŁADY KOMPUTEROWE
- UKŁADY KOMUNIKACYJNE I KOMERCYJNE

PRZEDSTAWICIELSTWO

- TRANSOPTORY
- WSKAŹNIKI ŚWIETLNE
- WYŚWIETLACZE LED
- PRODUKTY KODÓW KRESKOWYCH
- KONTROLERY I CZUJNIKI RUCHU
- TECHNIKA ŚWIATŁOWODOWA
- ELEMENTY W.CZ. I MIKROFALOWE
- PODZESPOŁY DO MONTAŻU POWIERZCHNIOWEGO (SMD)

DYSTRYBUCJA

- POTENCJOMETRY TRIMPOT
- HYBRYDY REZYSTOROWE
- REZYSTORY SUBMINIATUROWE
- BEZPIECZNIKI MULTIFUSE
- POTENCJOMETRY PRECYZYJNE
- POTENCJOMETRY PANELI CZOŁOWYCH I KODERY
- CEWKI I TRANSFORMATORY
- CZUJNIKI CIŚNIENIA, POŁOŻENIA I PRZYŚPIESZENIA

PMI

DYSTRYBUCJA

IBM Business Partner

meditronik sp. z o.o.

00-194 Warszawa,ul. Dzika 4
tel. (02) 6352263, 6352264
fax (02) 6352195, tlx 816075

SO/575/91

The Fourth International Conference Information Systems Development ISD'94

Katedra Organizacji Przetwarzania Danych Uniwersytetu Gdańskiego już po raz czwarty jest współorganizatorem Międzynarodowej Konferencji na temat: „Tworzenie systemów informatycznych". Tematyka ta cieszy się rosnącym zainteresowaniem, o czym świadczy ponad 40 referatów z 15 krajów zaprezentowanych na III Konferencji, zorganizowanej w Sopocie we wrześniu 1992 r. (Informatyka nr 5, 1993). Ta cykliczna, organizowana co dwa lata konferencja umiędzynarodowiła się również w sensie miejsca odbywania się: **IV Konferencja odbędzie się w alpejskim Bled, w Slovenii, w dniach 20–22 września 1994 r.** W skład Międzynarodowego Komitetu Programowego IV Konferencji wchodzą pracownicy naukowi i konsultanci z tej dziedziny z 20 krajów.

Celem konferencji jest tworzenie i utrzymywanie międzynarodowego forum wymiany wiedzy, doświadczenia i nowych propozycji stymulowania rozwoju dziedziny tworzenia systemów informatycznych (TSI). Przed konferencją będą organizowane seminaria prowadzone przez najwybitniejszych specjalistów z tej dziedziny. Wpisowe Konferencji wynosi 250 dol. USA.

Podstawowe tematy Konferencji:

- Teoretyczne podstawy systemów informatycznych;
- Nowe paradygmaty i trendy TSI;
- Metody i techniki TSI, narzędzia CASE;
- Systemy obiektowo zorientowane;
- Interfejsy użytkownika;
- Strategia informatyzacji;
- Planowanie systemów informacyjnych;
- Szkolenie profesjonalistów w dziedzinie TSI;
- Społeczne aspekty TSI;
- Wdrażanie systemów informatycznych w praktyce;
- Multimedia.

Zgłoszenia udziału można przesyłać, a także otrzymać wszelkie informacje pod adresem:

Prof. Jože Zupančič
School of Organizational Sciences
University of Maribor
64000 Kranj, Presernova 11, Slovenia
tel. +386 64 222 804, faks. +386 64 221424,
E-mail ISD @ FOV.UNI-MB.SI

Współprzewodniczący
Międzynarodowego Komitetu Programowego

Prof. dr hab. Stanisław Wrycza

Przypominamy!

Jedyną możliwością regularnego otrzymywania INFORMATYKI jest prenumerata.

Czasopismo można zaprenumerować w dowolnym czasie i na dowolny okres w Dziale Kolportażu Wydawnictwa SIGMA-NOT Spółka z o.o. **00-716 Warszawa, ul. Bartycka 20, skr. poczt. 1004, telefony: 40-30-86, 40-35-89, 40 lub 40-00-21 w. 293, 295, 299.**

Zaległe numery można zawsze kupić **w redakcji tel. 39-14-34**

dokumenty, drukującym czeki, rozpatrującym odwołania, prowadzącym politykę finansową, ustalającym składki od pracodawców, dokonującym operacji inwestycyjnych (obecnie znaczącą częścią funduszu są środki lokowane na giełdzie lub w nieruchomościach, a nie składki pracodawców). Odszkodowania muszą być wypłacane, a za nimi nie stoi przecież państwowa kiesa jako gwarant czy bufor.

Inne działanie WCB

W biurze głównym są również prowadzone różnorakie działania na rzecz poprawy bezpieczeństwa i higieny pracy w przemyśle. WCB zostało do tego zmuszone przez akt prawny, a pracodawcy, którzy składki muszą płacić, patrzą jednak na ręce WCB i ono z tego wykręcić się nie może. Ten rodzaj działalności jest odbierany dość dobrze, podobnie jak i działalność rehabilitacyjna. To chyba każdego dziwi, co mają wspólnego ubezpieczenia z rehabilitacją. Wymyślono to nie tak wiele lat temu i nowo wybudowane biuro głównie 50% powierzchni i 80% nakładów ma zainwestowane w ośrodek rehabilitacji zawodowej z hotelem, mnóstwem specjalistycznych urządzeń, sal, basenów, warsztatów z armią specjalistycznego personelu lekarskiego i rehabilitacyjnego. Wiele osób po wypadku odbywa tam rehabilitację lub trening w nowym zawodzie, bądź trening przywracający sprawność w nowym zawodzie, bądź trening przywracający sprawność w zawodzie dotychczas uprawianym. Jak wynika z różnych publikacji centrum rehabilitacji zawodowej jest jednym z najlepszych na świecie i płacącym składki ubezpieczenia pracowników to się opłaca.

Środki techniczne związane z przetwarzaniem

Centrum obliczeniowe dysponuje w obecnej chwili następującą platformą techniczną:

- sprzęt: IBM 3090-300J (zapowiadane jest powiększenie – *upgrade* – przez zakup procesora 9121-732) z 256 MB PAO, 32 kanałami, 150 GB pamięci dyskowej, I/O na kasetach taśm (3480-A22, B220), laserowe drukarki (jedna szybka), drukarka uderzeniowa (do druku czeków) oraz sieć terminali lokalnych w biurze głównym i biurach terenowych. W sumie do centralnego komputera jest podłączone ponad 1500 urządzeń, przez które użytkownicy i programiści, pielęgnujący i rozwijający system mają dostęp do zasobów. Budowę sieci w WCB zlecono wyspecjalizowanej jednostce, którą nie jest bezpośrednio przedsiębiorstwo telekomunikacyjne ale jedna z firm konsultacyjnych. W sieci są włączone IBM-owskie terminale (typu 3720) albo komputery IBM – PS/2, albo sieci LAN-y z takimi samymi komputerami.

- Oprogramowanie jest oparte na MVS/ES i CICS, nad którymi obecnie dominuje DB2 jako platforma podstawowa, również nadal w użyciu jest IDMS (baza danych o organizacji sieciowej). Natomiast narzędziami programisty są COBOL + SQL lub IDMS (dla oprogramowania wsadowego), a dla oprogramowania konwersacyjnego stosuje się generator dialogów ADSO (dla zastosowań z IDMS) lub CSP (dla zastosowań z DB2).

Dostęp do zasobów kontrolowany przez RACF

Wspomniane wyżej narzędzia są oczywiście tylko przykładem, stosuje się bowiem wiele dodatkowych pakietów (języków) wspomagania projektowania, programowania i testowania, a wśród nich takie jak OLQ – „ręczny navigator" w sieciowej bazie danych, QMF – podobny, ale o znacznie większej sile do operowania bazą relacyjną. Do niektórych celów stosuje się FOCUS – język IV generacji, w którym dość wygodnie można generować raporty z różnych baz danych. Na PC-tach używa się arkuszy kalkulacyjnych i oczywiście edytorów, ale obecny trend wypierania dużych komputerów powoduje duże zainteresowanie i udane eksperymenty z metodologią *klient-server* oraz pakietami umożliwiającymi budowę aplikacji w tej metodologii (np. Power-Bilder).

Jednym z najbardziej popularnych zastosowań o największej liczbie aktywnych użytkowników jest poczta elektroniczna (e-mail). Zastosowano tu pakiet MEMO Verimation (kupiony od Volvo i podobno popularny w Europie), prawdopodobnie wygodniejszy od znanego IBM-ego PROFS-a.

Struktura ISD

ISD jest pionem WCB na czele z wiceprezydentem skupiającym następujące cztery działy (na czele każdego z nich stoi dyrektor);

- Obsługi Klientów (*Client Services*);
- Ośrodek Obliczeniowy (sprzęt i przetwarzanie, albo jak tu się mówi – produkcja);
- Zarządzania danymi (administrator bazy danych);
- Obsługi administracyjno-technicznej.

Wewnętrzna struktura poszczególnych działów ma typową tematyczną organizację i tak np. dział *Client Services* skupia specjalistów projektujących i programujących skomputeryzowane zastosowania w grupach (wydziałach) odpowiadających zakresowi tematycznemu działów WCB:

- **FINANSE** – grupa projektująca (utrzymująca) systemy: odszkodowania jednorazowe, odszkodowania stałe (ang. *pensions*), składki pracodawców;
- **Odszkodowania** – systemy rejestracji roszczeń (nie koniecznie zakończonych odszkodowaniem) i statystyka odszkodowań;
- **Obsługa Medyczna** – systemy rejestracji działalności rehabilitacyjnej WCB, planowania zajęć z pacjentami i badanie efektywności rehabilitacji,
- **Prawny** – systemy odszkodowań za obrażenia wynikłe z czynów kryminalnych (WCB jest również upoważnione do obsługi tego typu odszkodowań – pracodawca zdecydował się na to, jak sądzę, ze względu na potencjał WCB i z merytorycznego podobieństwa z odszkodowaniami pracowniczymi, ponieważ łączy je wiele wspólnych pojęć, takich jak klasyfikacja urazów, ich weryfikacja, pojęcie odszkodowania jednorazowego czy dożywotniego itp., natomiast źródło finansowania odszkodowań jest zupełnie inne);
- **Bezpieczeństwo i Higiena Pracy** – systemy rejestracji kar nałożonych na pracodawców za niedostateczne bezpieczeństwo pracy w zakładzie czy na budowie (inspektorzy WCB mają uprawnienia do tego typu kontroli i jest ona stale prowadzona oraz systemy pomiarowe (zanieczyszczeń powietrza, wody, toksyczności itp):
- **Administracja** – systemy kadrowe (dla WCB), płace, poczta elektroniczna.

Client Services zawiera również grupy pracujące na rzecz tego działu – są nimi grupa technicznej jakości i grupa nowych technik. Pierwsza z nich pomaga projektującym w napotkanych problemach technicznych (język, pakiet, metoda, efektywność metody), a także kontroluje poprawność wykonywanych programów i projektów. Druga zaś śledzi ofertę rynkową oprogramowania (pakiety, narzędzia itp.) np. ostatnio dokonano wyboru pakietu CASE typu (Texas Instrument) oraz prowadzi pilotaż tego pakietu dla jednego z rozwijanych zastosowań).

Ośrodek Obliczeniowy

Ośrodek obliczeniowy składa się z następujących grup (wydziałów):

- Sterowanie produkcją – kilkuosobowa grupa organizuje planowanie prac wsadowych, tworzenie kopii archiwalnych, kopie takie wykonuje się dwa razy na dobę i przechowuje w co najmniej dwóch egzemplarzach (jedna z nich w specjalnych sejfach w oddalonym od ośrodka wyspecjalizowanym magazynie), kontroluje poprawność języka JCL wszystkich nowo wprowadzonych do produkcji prac, pełni rolę pierwszego wsparcia operatorów, gdy nastąpi awaria w trakcie nocnej lub popołudniowej zmiany (drugą linię wsparcia stanowi zwykle projektant danej pracy lub ktoś z administratora danych. O tym kogo wezwać, gdy nastąpi awaria, decyduje ustalona dla danej pracy procedura, lub gdy to nie jest jasne, dyżurna osoba z grupy sterowania produkcją);
- Operatorzy pracujący w systemie trzyzmianowym (po dwie osoby na jedną zmianę);
- Technicy, grupa liczy trzy osoby: dwóch techników, zajmujących się głównie instalacją sprzętu na stanowiskach pracy (w wyniku przeprowadzek lub zmiany rodzaju sprzętu) oraz szef do wstępnej oceny problemu technicznego; rzeczywisty serwis techniczny jest wykonywany przez producenta lub firmy wyspecjalizowane w danym sprzęcie (na marginesie ciekawostka – niesprawność techniczna systemu, to znaczy niemożność pracy z terminala podczas trzech lat obserwacji, zdarzyła się dwa razy – w sumie ok. 4 godzin);
- Programiści systemowi, ze specjalnościami: system MVS/XA, CICS, DB2 + VSAM + CSP, IDMS + inne pakiety, a w tym ENDEVOR (dla obsługi biblioteki kodu źródłowego we wszystkich językach stosowanych w WBC);
- Planowanie rozwoju – grupa ta oprócz planowania rozwoju sprzętu stanowi również grupę planowania perspektywicznego dla całego pionu ISD;
- Sterowanie bibliotekami – głównym zajęciem tej grupy jest badanie zużycia zasobów dyskowych, organizowanie ich odzyskania, akcje archiwowania, kontrola przechowywania kopii systemu;
- Sterowanie ochroną dostępu;
- Planowanie akcji „podniesienia po katastrofie" – grupa ta powstała nie tak dawno i zajmuje się planowaniem i organizowaniem niezbędnych działań, aby system WCB można było wznowić nie później niż po jednym dniu, jeśli został zniszczony w wyniku katastrofy. Grupa ta planuje stale działania w kooperacji z wyspecjalizowanymi firmami (ośrodki obliczeniowe rozrzucone w różnych regionach USA), które zapewniają działanie bieżące systemu, jeśli macierzysty ośrodek uległ katastrofie, np. takie jak trzęsienie ziemi (w zachodniej Kanadzie nie jest czymś wyjątkowym) – lub podobnej. Komórka ta została o dziwo dopiero zorganizowana po ostatnim trzęsieniu ziemi w San Francisco, poprzednie wypadki widać nie przemawiały dostatecznie do wyobraźni. Notabene nie był to pomysł samego WCB, ale akcja rządowa przeglądu budynków publicznych i ich przygotowania na trzęsienie ziemi.

Zarządzanie Danymi

Dział Zarządzania Danymi składa się z następujących grup (wydziałów):

- Projektowanie – modelowanie danych – zajmująca się projektowaniem modeli dla nowych zastosowań lub istotnych zmian w istniejących rozwiązaniach;
- Administracja danymi z podgrupami: bazy danych IDMS i bazy danych DB2; wykonują wszystkie czynności związane z bieżącym utrzymywaniem baz danych, takie jak odtwarzanie (kopiowanie z danych produkcyjnych) baz danych testowych po ich uszkodzeniu w wyniku testów, definiowanie (modyfikowanie) przeglądów dla DB2, modyfikowaniu uprawnień dostępu, a także dokonywaniem zmiany w danych (DB2) w wypadku, gdy dialog nie może tego dokonać (np. dane zostały uszkodzone i naprawa wymaga takiej zmiany, która przekracza zasady spójności, zdefiniowane dla dialogu, którym dysponuje użytkownik), a konstruowanie programu wsadowego, któryby to poprawił jest dużo bardziej pracochłonne i poprawek danych jest mało. Interesujące jest to, że żaden z użytkowników nie ma uprawnień do manipulowania danymi przy użyciu QMF – jest to silne narzędzie umożliwiające dowolną niemal akcję, a użytkownicy do manipulowania danymi dysponują jedynie dialogami skonstruowanymi w CSP, zgodnie z modelami funkcji i modelami danych (z zachowaniem zasad spójności i logiki danych);
- Osobną grupą jest zespół rozwijający narzędzia, którymi posługuje się Administracja Danych i wszyscy ci, którzy mają do czynienia z definiowaniem operacji na bazach danych. Zespół narzędzi nazwano niewyszukanie Workbench (warsztat), a zawiera on zespoły procedur wywoływanych dialogowo (dialogi ustalają również dla nich parametry) dla typowych pielęgnacyjnych operacji na bazie danych, a także dialogowo udostępniane biblioteki ze słownikami (z opisami funkcjonalnymi) danych istniejących w WCB.

Na marginesie uwaga o modelowaniu danych: WCB z dużym wysiłkiem doprowadziło, że wszystkie dane w systemach mają jednoznacznie zdefiniowany sens, opisana jest ich rola i nazewnictwo, nie udało się jednak, jak dotychczas, doprowadzić do tego, że wszystkie dane są dzielone (udostępniane) z jednego systemu (gdzie powstały) do innych, gdzie są używane. Nadal zdarza się, że ten sam typ danych jest kilkakrotnie wprowadzany do komputerowego środowiska WCB (czasem z różną wartością). Nie są to już obecnie dane o kluczowym znaczeniu dla systemów, choć wcale nie koniecznie drugorzędne – jak np. adres zamieszkania osoby zgłaszającej roszczenia o odszkodowanie (ang. *claimant*).

Dział Obsługi Administracyjno-Technicznej

Każdy zapewne sądzi, że skwituję ten dział stwierdzeniem podobnym do *co to koń, każdy wie*. Otóż nie, w pionie ISD dział ten pełni nie tylko rolę obsługi administracyjnej w zwykłym tego znaczeniu. Do obowiązków tego działu należy również tzw. kontrola jakości produkcji ISD. Ta działalność jest, jak mniemam, najmniej znana w organizacji przetwarzania danych w Polsce. Chcę poświęcić temu więcej niż wzmiankę. W istocie ta właśnie sprawa jest powodem napisania niniejszego artykułu.

Kontrola jakości

Miernik jakości zdefiniowano w oczywisty i niewymyślny sposób: jakość jest tym większa, im klient (odbiorca produktu, usługi) jest bardziej usatysfakcjonowany. Proste rozumowanie prowadzi do wniosku, że z przetwarzania danych klient (odbiorca wyników) jest zadowolony wtedy, jeśli wszystko przebiega bez zakłóceń (to jest minimum satysfakcji) a powyżej są specjalne pozytywne zdarzenia i tymi jako nadzwyczajnymi kontrola jakości nie zajmuje się.

Satysfakcja użytkownika równie dobrze mierzy wynik pracy projektujących zespołów, jak i potem funkcjonowanie systemu. I nie jest ważne, jaką metodą projektujący osiągnęli satysfakcję użytkownika, ważne jest aby nie było negatywnych reakcji, a co najmniej – nie było zakłóceń. Przyjrzyjmy się dokładniej temu, który jest użytkownikiem w poszczególnych fazach cyklu projektu i kto może „marudzić":

- faza pierwsza – *bussnes oportunity*, inaczej mówiąc koncepcja wstępna, gdzie identyfikowane są skutki ekonomiczne zastosowania rozwiązania komputerowego; oczywiste jest, że odbiorcą wyniku tej fazy projektowania jest szefostwo firmy, dla której jest wykonywany projekt,
- druga faza – *bussnes model* jest zatwierdzany ponownie przez kierownictwo na podstawie analizy i opinii wykonanych przez operacyjny szczebel przedsiębiorstwa,
- faza trzecia – *functional model* – odpowiednik projektu technicznego, którego odbiorcami są: kierownictwo szczebla operacyjnego, a ponadto odbiorcy (użytkownicy) z samego ISD w postaci zespołów programujących i testujących (zawsze dla większego projektu test funkcjonalny jest prowadzony przez osobny zespół, a nie przez zespół autorski);
- faza czwarta – *implementation* – tu po stronie nie związanej z ISD występuje już pełna hierarchia przedsiębiorstwa – od najwyższego szczebla, do poszczególnych stanowisk, których implementacja dotyczy, a po stronie ISD, poza zespołami projektującymi, dochodzą te od kontroli i sterowania produkcją (sterujący harmonogramowaniem i poprawnością produkcji, operatorzy, kontrola we-wy), a także grupa wsparcia systemowego;
- faza piąta – *post implementation review* – tu, podobnie jak i w poprzedniej fazie, odbiorcami (użytkownikami) są wszyscy wymienieni w poprzednich fazach plus sami projektujący;
- faza szósta (ostatnia w cyklu) – *maintenance* – to stała pielęgnacja systemu (z której może powstać *bussnes oportunity report*); faza ta przebiega najlepiej, jeśli niemal wszyscy o jej istnieniu zapominają, tzn. użytkownicy nie mają zastrzeżeń do funkcjonowania systemu (pracownicy firmy są usatysfakcjonowani z wyników, kontrola produkcji nie wnosi zastrzeżeń, kontrola zużycia zasobów nie postuluje zmian).

Powyższe, jak sądzę oczywiste stwierdzenia, nie rzucają jeszcze pełnego światła na funkcjonowanie kontroli jakości w ISD. Wychodząc z praktyki, która mówi, że najbardziej bulwersujące są nagłe zdarzenia, a z tej kategorii tzw. *abendy* (*abend* w produkcji oznacza wadę systemu), wywołują najwięcej zainteresowania i niepokoju użytkowników, kontrola jakości stara się tak działać, aby w fazie produkcji unikać abendów. Oczywiście nieudane próby zakończenia faz projektowych innych niż ostatnia, powodują również abendy, ale innego rodzaju, nie koniecznie kończące się zwolnieniem szefa projektu, ale napewno zaszarganiem jego opinii. Do tych faz kontrola jakości praktycznie nie wtrąca się bezpośrednio, jedynie nadzorowane przez nią standardy i opracowane raporty o nieudanych przedsięwzięciach mają wpływ na te fazy. Do niedawna jedyną metodą unikania niepowodzenia w fazie produkcji (*maintenance*) były rozwiązania organizacyjne; ostatnio modne są różnorakie narzędzia reklamujące się jako *abend free*. Nie chodzi zwykle (przynajmniej w fazie produkcji) o abendy spowodowane błędami semi formalnymi, a błędami logiki zastosowania, a tych żadne narzędzie samo z siebie nie wyeliminuje i nie zrobią tego również CASE tools! Natomiast stałe stosowane rozwiązania organizacyjne to:
- jakakolwiek zmiana w środowisku produkcyjnym wymaga specjalnej procedury, a zmianą tą może być np. nowy lub poprawiony program, jakakolwiek zmiana w oprogramowaniu systemowym, jakakolwiek zmiana w funkcjonujących procedurach, jakakolwiek zmiana w funkcjonującej dokumentacji, zmiana danych dokonana nie przez użytkownika przy użyciu standardowo funkcjonujących operacji, a przez administratora bazy;
- standard kodowania, dostępu (używania poszczególnych komend), nazewnictwa, organizowania bibliotek i zbiorów;
- standard ról i uprawnień np. nie ma mowy, aby osoba z grupy sterowania produkcją – posiadająca najwyższe uprawnienia – dokonała jakiejkolwiek operacji na danych lub bibliotekach, podobnie ma się rzecz z grupą technicznego wsparcia: ludzie z tej grupy – programiści systemowi w swoim zakresie mają uprawnienia do wszystkich zasobów maszyny, natomiast w środowisku produkcyjnym zachowują się tak, jak programiści aplikacyjni;
- procedura informowania o zdarzeniach (oprogramowanie utrzymujące bazę danych zdarzeń – powszechnie stosowanym systemem, przynajmniej na północy Ameryki, jest INFOMEN).

Jako standard przyjęto notowanie następujących zdarzeń: abendów w sferze produkcji, jakiekolwiek kłopoty ze sprzętem lub oprogramowaniem systemowym, a także wszystkie zmiany w produkcji.Osoba wprowadzająca zapis do INFOMEN-a wypełnia szereg standardowych rubryk (bez ich wypełnienia system się „nie odczepi"). W INFOMEN-ie można co najwyżej zaznaczyć nieaktualność dokumentu z podaniem powodu (nie można niczego z IFOMEN-a wykreślić!). A jako dalszy ciąg standardu – wszyscy kierownicy w ISD mają obowiązek co najmniej raz dziennie przejrzeć INFOMEN, zwłaszcza nowo opisane problemy i jeśli dotyczą one jego obszaru kompetencji, podjąć niezbędne działania. W zasadzie każdy pracownik ISD przegląda INFOMEN, a więc błędy stają się informacjami publicznymi i jest to rodzaj presji, który dość skutecznie wpływa na samokontrolę i poszukiwanie najskuteczniejszych metod poprawnego promowania do produkcji. Natomiast jakiekolwiek zmiany w produkcji wymagają następujących zabiegów autora (wykonawcy):

- Wypełnienie w INFOMEN-ie dokumentu „promocji", czyli zmiany w środowisku produkcyjnym.

- Zatwierdzenia zmiany przez grupę wyznaczonych osób (najczęściej przedstawiciele istotnych dla biegu produkcji komórek); jeśli jest to niewielka zmiana robi się to na bieżąco przeglądając INFOMEN (są tam notowane zatwierdzenia), jeśli zaś poważniejsza, to na odbywanym trzy razy w tygodniu o tej samej porze krótkim posiedzeniu (*change meeting*), gdzie autor promocji ma obowiązek uczestniczyć w zebraniu i odpowiadać na ewentualne pytania; nadmiar pytań odrzuca promocję jako nie przygotowaną. Inaczej mówiąc w interesie autora jest wcześniejsze przedyskutowanie możliwie ze wszystkimi zainteresowanymi proponowanej zmiany i uzgodnienie z nimi wszystkich szczegółów, w przeciwnym razie bowiem *change meeting*, promocji nie zatwierdzi i zebranie trzeba będzie powtórzyć a każde „spalone podejście" jest notowane w INFOMEN-ie.

- Po zatwierdzeniu promocji odbywa się procedura promocyjna tzn. przemieszczenie elementu z bibliotek projektowych do bibliotek produkcyjnych; operacji tych dokonują tylko wyznaczone osoby.

- Gdyby po promocji nastąpił abend, to nie tylko notuje się ten fakt jako zdarzenie w produkcji, ale kontroler zmian (osoba stale doglądająca bazy INFOMEN-a) ma obowiązek uaktualnić odpowiednie zapisy, aby połączyć zaistniały abent z promocją, która staje się wówczas nieudana i jej autor już teraz nadzorowany nie tylko przez szefa (o abendzie już zdążył się on dowiedzieć), ale i przez inną osobę (zwykle ta sama, która dogląda INFOMEN-a), aby naprawić nieudaną promocję (naprawić abend produkcyjny).

- Do dobrego obyczaju należy przegląd działania promowanego elementu (po promocji w produkcji) przez samego autora; gdy promocja była nieudana i fakt ten był zanotowany – przegląd po promocji (po naprawie) i akceptacja użytkownika są już wymagane. Akceptacja ta ma charakter deklaracji autora promocji, ale to absolutnie wystarczy.

dokończenie na s. 19

Piąta Jubileuszowa Międzynarodowa Górska Szkoła PTI Szczyrk
21-25.06.1993 r.

STANISŁAW KĘDZIERSKI
Katedra Informatyki
Akademia Ekonomiczna
Katowice

Modelowanie obiektów dynamicznych

Pojęcie obiektu pojawiło się w latach siedemdziesiątych w wyniku prac nad nowymi metodami programowania. Booch [1] wprowadza klasyfikację języków programowania według kryterium abstrakcji, którymi się one posługują:
- generacja pierwsza (1954–1958) – abstrakcja matematyczna,
- generacja druga (1959–1961) – abstrakcja algorytmiczna,
- generacja trzecia (1962–1970) – abstrakcja danych,
- luka generacyjna (1970–1980),
- *N*-ta generacja (1980 – dotychczas) abstrakcja zorientowana na obiekty.

Programy są jednym z wielu elementów systemu informatycznego (SI) i stanowią rezultat procesu projektowania. Projektowanie zaś jest niczym innym jak modelowaniem istniejącej rzeczywistości. Z punktu widzenia projektanta interesująca jest część otaczającego świata nazywana **dziedziną przedmiotową**. Dziedzina przedmiotowa (ang. UoD – *Universe of Discourse*) to zbiór wszystkich możliwych jednostek, które aktualnie leżą w zasięgu zainteresowania odpowiedzialnych osób (projektanci, analitycy).

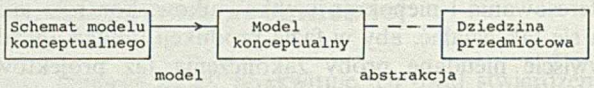
Model konceptualny dziedziny przedmiotowej to pewna abstrakcyjna jednostka, w której jest zawarte wspólne pojmowanie dziedziny przedmiotowej przez projektantów. Model konceptualny powstaje z dziedziny przedmiotowej jako rezultat procesu **modelowania konceptualnego** (występującego też pod nazwą analizy informacyjnej). Projektant powinien być zainteresowany tym, aby doprowadzić ten proces do takiej postaci (przy użyciu metod formalnych), by mógł on stanowić podstawę do automatycznego generowania rozwiązania.

Modele konceptualne obiektowo-zorientowane

W modelowaniu konceptualnym należy odciąć się od modeli danych, które są zorientowane na komputer (opisując rozmieszczenie danych w mc.). Należy rozpatrywać dwa rodzaje jednostek: obiekty i masy. Obiekt to cokolwiek, co może być pomieszczone w zbiorze (w sensie matematycznym). Zawsze mówi się o zbiorze (aktualnym lub możliwym) ludzi, przedsiębiorstw itp., znaczy to, że obiekty mogą być policzone. Zbiór obiektów nie musi być przeliczony, ale ma swą moc. Z drugiej strony masa nie może być policzona, czyli nie może być elementem zbioru. Za przykłady mas mogą służyć: woda, zysk, waga itp. O masy można się pytać: jak dużo? ale nie konkretnie: ile? Zwrócić należy uwagę, że masy nigdy nie są reprezentowane w bazach danych czy modelach konceptualnych. Dla mas (o których powinny być przechowywane jakieś informacje) definiowana jest funkcja miary z masy do obiektu dyskretnego: np. zysk w jednostkach pieniężnych, waga w kilogramach.

Obiekty charakteryzują się posiadaniem atrybutów. Wiele obiektów może mieć te same własności, przy czym nie są to te same obiekty, np.: książki w bibliotece, samochody opuszczające fabrykę. Są one takie same (autor, tytuł, treść; marka, kolor, wyposażenie) a różni je tylko numer (sygnatury, lub fabryczny). Każdy obiekt musi mieć unikalny globalny identyfikator, celem umożliwienia ich rozróżnienia. Rozpatrując identyfikatory obiektów należałoby bardziej wnikliwie skoncentrować się na ich cechach:
- Obiekt ma swój identyfikator podczas całego czasu istnienia, niezależnie od stanu w jakim znajduje się dziedzina przedmiotowa;
- Identyfikator obiektu jest pewnego określonego typu. Licząc obiekty liczymy identyfikatory np.: możemy przyjąć, że zbiór pasażerów autobusu był trzyelementowy, mimo że ta sama osoba trzy razy korzystała z przejazdu. Zawsze mówimy, że był nie: „50", lecz „50 pasażerów" czy też: „50 osób" (pasażer lub osoba jest tutaj typem);
- Identyfikatory mogą być strukturalnie zależne od innych identyfikatorów, np.: dodanie nowej osoby do zbioru może zmienić wartości pól *średni wiek* czy też *największy wiek*.

Zorientowany obiektowo model konceptualny wytwarzany jest w procesie modelowania. Można powiedzieć, że każdy model konceptualny jest pewną abstrakcją dziedziny przedmiotowej i można patrzeć na model konceptualny jako na schemat konceptualny, przedstawia to rys. 1.



Rys. 1. Modelowanie konceptualne

Schemat modelu konceptualnego (SMK) powinien mieć formalną postać (żeby można było formalnie opisać relacje pomiędzy SMK i MK). Przykładowo chcemy, aby SMK był teorią w formalnym języku z MK jakoł modelem.

Metodyki projektowania proponują różne języki specyfikacji schematu modelu konceptualnego oraz pewien zbiór struktur, który może być reprezentowany w MK, a także grupę porad praktycznych wyjaśniających w jaki sposób z wiedzy analityka o dziedzinie przedmiotowej wytworzyć specyfikację MK. Porady praktyczne to głównie zbiór kolejnych zadań do wykonania, co w klasycznych metodykach przyjmowało formę liniowego następstwa. Zadania sprecyzowane były aż do poziomu żądanych detali (projekty zbiorów, wydawnictw, algorytmów programów).

Metodyka może być obiektowo zorientowana co oznacza że modeluje ona rzeczywistość w sposób uwypuklający obiekty. Istnieją trzy powody wyboru takiego podejścia:
- orientacja na dziedzinę przedmiotową umożliwia zdobycie dostatecznej wiedzy o niej i uniknięcie błędów w dalszych etapach,
- umożliwienie identyfikacji obiektów, ich klasyfikacja i dynamika (Booch uważa, że koncentracja na obiektach z DP koresponduje z zasadą modularyzacji software'u),

- specyfikacja modelu przed specyfikowaniem funkcjonalnym, czyli powodowanie większej stabilności systemu informatycznego.

Klasyfikacja obiektów

Obiekty, charakteryzują się istotnymi i ciekawymi cechami. Wymienienie ich wydaje się bardzo pouczające:
- obiekty są dyskretne,
- operacje łączenia lub rozdzielania mas dają masę tego samego typu,
- każdy obiekt musi mieć globalny unikalny identyfikator.

Obiekty z dziedziny przedmiotowej można pogrupować w klasy. W celu wskazania obiektu, wskazuje się konkretne wystąpienie w jednej lub większej liczbie **klas** obiektów, jak np.: człowiek, zatrudniony, samochód itp.

Klasy (uniwersa) z filozoficznego punktu widzenia charakteryzują się dwiema cechami:
- Typ (klasa) orzeka o rzeczy, np. *czerwony* jest typem ponieważ mówi, że coś jest czerwone. Typ może też być egzemplarzem innego typu, np. *czerwony* jest wystąpieniem typu *kolor*. Typy mają wystąpienia poszczególne zaś obiekty reprezentują tylko siebie, np.: Warszawa nie jest typem ponieważ orzeka tylko o sobie.
- Typy orzekają o *wielu* rzeczach, czyli o tym co jest wspólne dla pewnej liczby elementów.

Wszystkie obiekty należące do danej klasy mają tę samą strukturę. Może być wiele wystąpień o wielu atrybutach w danej klasie. Umysł analityka ogranicza liczbę atrybutów. Można powiedzieć, że atrybuty określają w pewnym sensie klasę. Klasę można scharakteryzować przez podanie jej właściwości.

Pierwszą własnością jest to, że klasa powinna być naturalna, czyli że w prosty sposób można ją łączyć lub dzielić i określić przynależność obiektu do klasy. Używając pojęć logiki można powiedzieć, że klasą naturalną jest największy zbiór wystąpień, dla którego prawdziwe są warunki przynależności do klasy. Przykładowo, zbiór osób w pewnej dziedzinie przedmiotowej może być największym zbiorem wystąpień określonych warunkami: posiadanie nazwiska oraz wiek < 120 lat.

Drugą własność klasy naturalnej stanowi fakt, iż istnieje ona w swojej naturze. Analityk może znaleźć informacje o niej w DP.

Trzecią cechą klasy naturalnej jest jej podstawowość lub warunkowość. Podstawowość oznacza niezmienianie cech podczas przechodzenia DP przez różne stany, np.: człowiek nie może istnieć jako wystąpienie nie będąc elementem klasy *osoba*. Warunkowość zaś oznacza możliwość przestania bycia elementem danej klasy, jeśli zmienił się stan DP, np.: można przestać być elementem klasy *student* czy też *pracownik*. Często mówi się, że obiekt z klasy warunkowej gra *rolę*. Aby móc grać rolę obiekt musi należeć do klasy naturalnej podstawowej, np.: osoba może grać rolę pracownika, studenta, czytelnika itp. Wynikają stąd wnioski mówiące o tym, że:
- rola musi być podklasą klasy naturalnej,
- żadna klasa naturalna nie może być podklasą roli.

Obiekt ma strukturę, która może być zależnie od okoliczności podstawowa lub warunkowa. Wszystkie podstawowe struktury obiektu wyrażone są przez identyfikator. Struktura warunkowa może zmieniać się w czasie, np.: zbiór pracowników wydziału, wiek osób, kolejność występowania są reprezentowane przez wartości atrybutów co można sformalizować następująco:

zatr $:WYDZ \rightarrow OSOBA$

wiek $:OSOBA \rightarrow NAT$

max-wiek $:OSOBY \rightarrow NAT$

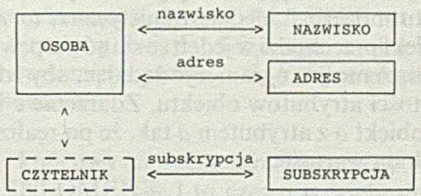
gdzie:
WYDZ, OSOBA, OSOBY są odpowiednio typami identyfikatorów wydziałów, osób i skończonych zbiorów osób.

Interesujące z punktu widzenia możliwości automatyzacji dalszych prac nad projektem systemu informatycznego jest formalne zapisanie specyfikacji tego systemu. Jedną z propozycji jest CMSL (ang. *Conceptual Modelling Specification Language* [3]). Jako przykład może posłużyć definiowanie obiektów dotyczących czytelników:

```
   import
     OsobaIde, Nazwisko, Adresy
   identifier sorts
     OSOBA
   attributes
     nazwisko :OSOBA   -> NAZWISKO
     adres    :OSOBA   -> ADRES
end spec OsobaAtrybuty


object spec CzytelnikAtrybuty
   import
     Osoba, Boolean,
              ZbiórSubskrybentów
   roles
     CZYTELNIK < OSOBA
   attributes
     subskrypcja :CZYTELNIK  ->
                        SUBSKRYPCJA
end spec CzytelnikAtrybuty
```

Powyższe dwa opisy mają swoją graficzną reprezentację przy wykorzystaniu grafów skierowanych (rys. 2).



Rys. 2. Struktura semantyczna osób i czytelników

Niezaetykietowana strzałka na rys. 2 oznacza funkcję identyczności, czyli obrazuje to, że *CZYTELNIK* jest podklasą *OSOBY*. Górna część rysunku przedstawia prostą hierarchię mówiącą o tym, że osoba reprezentuje połączenie nazwiska z adresem. Przerywaną linią oznaczona jest rola jaką może pełnić osoba.

W opisie specyfikacji *OsobaAtrybuty* importowane były trzy specyfikacje, przy czym milcząco założono, że są one już opisane. I tak *OsobaIde* określa identyfikatory typu *OSOBA*:

$p_0, p_1, p_2, p_3, \ldots$

Na tym typie określona jest funkcja sprawdzająca równość dwu identyfikatorów:

eg: $OSOBA * OSOBA \rightarrow BOOL$

OsobaIde oraz *Nazwisko* określają **abstrakcyjne typy danych** identyfikatorów osób i ciągów znaków. Adresy specyfikują abstrakcyjne typy obiektów tak jak czynią to *OsobaAtrybuty*.

Tak więc każdą osobę można traktować jako trójkę o przykładowej postaci:

$< p_1 <$ *nazwisko: 'Kowalski';*
*adres: a*15 $\gg$.

CzytelnikAtrybuty deklaruje rolę *CZYTELNIKA* dla osób oraz zbiór subskrypcji czytelnika. Notacja:

CZYTELNIK < OSOBA

oznacza, że klasa czytelników zawiera się w klasie osób, przy czym jest możliwe, aby klasa *CZYTELNIK* była pusta. Formalny zapis bycia czytelnikiem przybiera postać:

$< p_1 <$ *nazwisko: 'Kowalski',*
*adres: a*15,
subskrypcje: $\{s_1, ..., s_n\} \gg$
gdzie $s_1, ..., s_n$ są identyfikatorami subskrypcji.

Specyfika zdarzeń

Obiekty mogą pełnić pewne role, ale aby do tego doszło muszą zaistnieć jakieś zdarzenia. Zdarzenia potrzebne są do modelowania zmian zachodzących w DP. Ogólnie rzecz biorąc można wyróżnić dwa rodzaje zdarzeń:
- zmiana roli,
- zmiana wartości atrybutu.

Mówiąc o zdarzeniu ma się na myśli to, że obiekt wykonuje je lub że podlega jego działaniu. Ponieważ obiekty mają swe identyfikatory więc można powiązać zdarzenia z konkretnymi obiektami. Pomiędzy obiektami o_1 i o_2 zachodzi interakcja, jeśli obserwują one swe działania. Rozumieć to należy następująco: istnieje co najmniej jeden obiekt (dla innych może być to niedostępne), który obserwuje pełnienie ról lub zmianę atrybutów innego obiektu. Każde zdarzenie może zmienić stan obiektu (oczywiście z wyjątkiem jego identyfikatora). Zmiana stanu obiektu może być niedeterministyczna, to znaczy, że ze stanu *st*1 obiekt przechodzi w jeden ze stanów z pewnego zbioru. Niedeterminizm można ograniczyć żądając, aby zdarzenie było funkcją wartości atrybutów obiektu. Zdarzenie *e* jest wykonywane przez obiekt *o* z atrybutem *a* tak, że po realizacji *a* będzie miało określoną wartość.

Jak widać istnieje związek między zmianą stanu a zdarzeniem. Jeśli obiekt zmienia stan – wykonuje on zdarzenie. W drugą stronę to stwierdzenie niekoniecznie musi być prawdziwe, np.: po wykonaniu zdarzenia obiekt może pozostać w tym samym stanie. Obserwator tego zdarzenia mógł zmienić swój stan, czyli można powiedzieć, iż *coś* się zdarzyło. Ogólnie wystąpienie zdarzenia zmienia stan *gdzieś* w DP lub w obiekcie wykonującym zdarzenie lub u obserwatora. Kiedy już wiadomo, że nie było zmian w DP to można powiedzieć, że nie wystąpiło żadne zdarzenie.

Zdarzenia zmieniające stan pojedynczego obiektu są zdarzeniami lokalnymi. Zdarzenia takie nie zmieniają już nic więcej w DP. Jeśli istnieje powiązanie wykonania zdarzenia z obiektem to można to rozszerzyć na klasę obiektów. Przykładowo, jeśli zdarzenie *zwiększ-wiek* (*p*) jest wykonywane przez osobę, to *zwiększ-wiek* jest zdeklarowane dla wszystkich możliwych osób (z klasy *OSOBA*).

Inną ważną cechą zdarzeń jest ich atomowość, czyli spełnienie dwu warunków.
- zdarzenie nie trwa dłużej niż pewien okres czasu,
- zdarzenie nie składa się z podzdarzeń

co prowadzi do wniosku, że wykonywanie zdarzeń nie ma stanów pośrednich.

Obiekty powinny móc komunikować się między sobą co można modelować jako wspólną akcję dwu obiektów, które dzielą się zdarzeniem. Formalny zapis **enkapsulacji i funkcji komunikacji** jest następujący:

$e_1 | e_2 = e_3$

gdzie: | oznacza konieczność komunikowania się zdarzenia e_1 ze zdarzeniem e_2, i że zdarzenie e_3 składa się z synchronicznych wykonań e_1 i e_2. Zdarzenie e_3 zachowuje atomowość ponieważ nie ma stanów pośrednich. Przykładem tej definicji jest zdarzenie:

zwiększ-stanowisko |
zwiększ-pensję = awans.

Zdarzenia zenkapsulowane nazywane są **komunikatami** (ang. *messages*). Enkapsulacja i komunikacja odbywa się na dwu poziomach: lokalnym i globalnym. Komunikat lokalny występuje tylko wtedy, gdy wykonywane są synchronicznie co najmniej dwa zdarzenia na tym samym obiekcie; globalny – gdy wykonywane są synchroniczne zdarzenia na różnych obiektach. Analogicznie określana jest komunikacja lokalna i globalna.

W komunikowaniu się obiektów obowiązują następujące zasady:
- zdarzenia wykonywane przez pojedynczy obiekt wykluczają się wzajemnie,
- dowolny zbiór zdarzeń bez *message* wykonywany przez różne obiekty może być synchroniczny,
- efekt komunikacji wykonywanej przez różne obiekty na obiektach to wykonujących jest równy efektowi uczestniczących zdarzeń na tych obiektach.

Jako przykład specyfikacji zdarzeń opisane będą: zmiana adresu, subskrypcja i udział w konkursie.

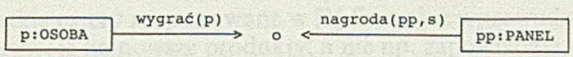
```
object spec OsobaZdarzenie
  import
    OsobaAtrybuty
  events
    zmiana-adresu:
        OSOBA×ADRES   > OSOBA
  variables
    p: OSOBA
    a: ADRES
  local event constrains
    adres(
        zmiana-adresu(p,a))=a
end spec OsobaZdarzenie
```

```
object spec CzytelnikZdarzenie
  import
    CzytelnikAtrybuty
  events
    subskrypcja:OSOBA*subskrypcja  -> CZYTELNIK global message
    dołączanie :OSOBA*subskrypcja  -> CZYTELNIK global message
    wejście    :OSOBA*subskrypcja  -> CZYTELNIK global message
    wygrać     :CZYTELNIK  > CZYTELNIK global message
  variables
    p:OSOBA
    s:subskrypcja
  local event constrains
    subskrybowanie(subskrypcja(p,s))={s}
    subskrybowanie(subskrypcja(p,s))=subskrybowanie(p)+{s}
end spec CzytelnikZdarzenie
```

```
conceptual model spec Tytuł
  import
    CzytelnikZdarzenie, Panel, Skrzynka
  variables
    pp:PANEL
    p :OSOBA
    s :subskrypcja
    sk:SKRYTKA
  global communications
    załóż-SUBSKRYPCJA(db)|subskrypcja(p,s)
    załóż-SUBSKRYPCJA(db)|dołączanie(p,s)
    nagroda(pp,s)|wygrać(p)
    wejście(p,s)|otrzymać(m,s)
end spec Tytuł
```

W drugim wierszu od dołu jest: otrzymać (m,s); powinno być: otrzymać (sk,s)

OsobaZdarzenie opisuje jedno zdarzenie dla osoby a mianowicie zmianę adresu. *CzytelnikZdarzenie* podaje możliwość prenumeraty i stwarza osobie możliwość grania roli czytelnika i powiększania zbioru prenumerat. Każdy czytelnik może wziąć udział w konkursach z szansą na wygraną. W modelu konceptualnym *Tytuł* importowane są specyfikacje odpowiednich obiektów oraz dodawane informacje globalne a w szczególności specyfikacja komunikacji globalnej. Komunikacja w modelu konceptualnym przedstawiona jest za pomocą diagramu komunikacyjnego (rys. 3).



Rys. 3. Diagram komunikacyjny

W artykule zostały poruszone podstawowe problemy obiektowo zorientowanego modelowania konceptualnego systemów informatycznych. Ten sposób pojmowania i modelowania rzeczywistości (DP) wnosi nowe wartości do pracy analityka i projektanta. Nie będzie ona wolna od błędów, ale z pewnością zdecydowanie zmniejszy się ich liczbę.

Należy spodziewać się, że dalszy rozwój metod informatycznych będzie wiązał się z ich formalizacją, co przyczyni się do automatyzacji prac. Omówiony w niniejszej pracy język CMSL jest jednym z przykładów takiego sformalizowanego podejścia.

LITERATURA

[1] Booch G.: Object-oriented development. IEEE Transactions on Software Engineering, 1986, pp. 211–221

[2] Cameron J.R.: An overview of JSD, IEEE Transaction on Software Engineering, 1986, pp. 222–240

[3] Wieringa R.: Steps towards a method for the formal modeling of dynamic objects, Data & Knowledge Engineering, 1991, pp. 509–540.

Ośrodek obliczeniowy firmy ubezpieczeniowej w Kanadzie

dokończenie ze s. 15

Na pierwszy rzut oka procedura promocyjna jest pełna biurokratycznych barier i skomplikowana, wrażenie to pogłębia fakt, że za promocję ważną, wymagającą fizycznej obecności autora na *change meeting* uważa się taką zmianę, gdy proponowany program ma komendę *UPDATE* (albo *DELETE* lub *STORE*), i to nie zależnie od tego, czy poprawka dotyczyła w jakiś sposób tych komend, czy też nie (mogła mieć charakter czysto kosmetyczny), a mimo to zmiana jest poważna, bo dokonana na poważnym (*UPDATE*) programie. Jedynie programy raportujące mogą być zatwierdzone na bieżąco przez „kontrolerów zmian" (stali uczestnicy *change meetingu* z uprawnieniem podpisu zmiany) choć i wówczas zdarza się, że mają pytania do autora i zmiana jest zawieszana (odraczana). Aby uniknąć niepowodzenia podczas zatwierdzania promocji (na zebraniu czy na bieżąco), wprawny (doświadczony) autor w dokumencie promocyjnym dość krótko, ale unikając niejasności, opisuje dokonaną zmianę tak, aby kontrolerzy mogli bez pytania podjąć decyzję dopuszczenia zmiany do produkcji. Opisywana powyżej procedura promowania dotyczy promocji związanych z fazą *maintenace*. Nieco odmiennie przebiega promowanie do produkcji większych całości (podsystemów) jako wynik projektowania lub implementacji projektu.

Implementowanie systemu

Jak już wcześniej wspomniano, system jest testowany przez osobny zespół na podstawie szczegółowego planu testowania i raportowania testu. Plan testowania i sam proces testowania mają własną szkołę i jest niemal znormalizowany, przynajmniej jeśli chodzi o ciąg czynności i powstające podczas testu dokumenty. Najistotniejszymi dokumentami testu są tzw. protokoły rozbieżności (ang. *discrepancies*), gdzie odnotowuje się najdrobniejsze odchylenia w funkcjonowaniu elementu (programu lub dialogu) w stosunku do projektu.

Dokonane poprawki i brak dokumentu rozbieżności kwalifikuje system do tzw. testu funkcjonalnego, tego dokonują już użytkownicy i znowu powstają dokumenty rozbieżności. Nad rozwojem projektu stale czuwa komitet sterujący (szefem jest główny użytkownik) i on decyduje, czy wyniki testu funkcjonalnego kwalifikują system do promocji oraz jakie rozbieżności powinny być usunięte jeszcze przed promocją, a jakie mogą poczekać na fazę *maintenance*.

Dokonuje się znanego chwytu *RELEASE no. n*, bez tego systemy nie ujrzałyby światła dziennego, a dzięki kolejnym *releason* z powodzeniem promuje się kolejne ulepszone wersje i nowe zastosowania.

Oczywiście końcowy moment samej promocji systemu jest notowany w INFOMEN-ie i to często w wielu dokumentach (promocja programów wsadowych, programów dialogowych, instrukcji, dokumentacji i – być może – specjalnych procedur) np. promocja jest dokonywana w weekend i potrzebni są autorzy programów w ustalonych godzinach pod telefonem, a wsparcie systemowe na miejscu w ośrodku itp. Standard ról wymaga dokładnego i przemyślanego planu promocji systemu, na ogół jest to duże przedsięwzięcie.

* * *

Czy opisana tu organizacja pracy jest idealna? Widoczna jest szczególnie jej nieoszczędność – ale może to jest warunek niezawodności? Tzw. bezpośrednio produkcyjny personel ISD (analitycy i programiści związani bezpośrednio z projektowaniem i utrzymywaniem systemów + sterowanie produkcją), to nie więcej niż 40% zatrudnionych – reszta to wsparcie, kontrola, standard, planowanie itp., itd. Mimo, że nie ma ani widocznego nacisku na pracowników, ani pośpiechu, wszystko razem działa i tylko czasami dochodzą pomruki zapowiadające reorganizację (zapewne ktoś szykuje się do objęcia jakiejś upatrzonej pozycji), ale i ona nie burzy form pracy, czy standardów, bo reorganizacje duże czy małe są dokonywane niemal stale, a ich efekty, to zmiany priorytetów, nie metod pracy.

MAREK MIŁOSZ
Katedra Informatyki
Politechnika Lubelska

Quo vadis czyli ciąg dalszy o wyborze relacyjnej bazy danych

Dystrybutorzy oprogramowania i osoby z nimi ściśle współpracujące bardzo czule podchodzą do wszystkich spraw, związanych z dostarczanym przez nich produktem. Jest to zrozumiałe z wielu powodów i tego podstawowego także. W ich mniemaniu oferowane przez nich systemy są najlepsze, najnowocześniejsze, najbezpieczniejsze, najlepiej się sprzedają (lub też mają najlepszą dynamikę sprzedaży), mają najlepszy serwis posprzedażny, największy rynek (lub chociażby jego wycinek), są używane przez najbardziej godne zaufania instytucje (np. Kongres Stanów Zjednoczonych, CIA lub Pentagon). Słowem: samo NAJ. Jeśli porówna się oferty konkurencyjnych wyrobów, to okazuje się, że nie jest to do końca prawdą.

Który z programów wybrać? Czym kierować się podczas wyboru? W jaki sposób dokonać oceny? Które parametry programu są najważniejsze, a które mniej ważne?
Jednym słowem: Quo vadis projektancie i przyszły użytkowniku systemów informatycznych?

W jakże trudnej sytuacji znajduje się przyszły użytkownik, który doczyta się, że ten przysłowiowy już Pentagon używa wielu konkurencyjnych produktów jednocześnie, że w Europie najlepiej sprzedaje się program *X*, ale już w Stanach Zjednoczonych – program *Y*. Jeśli nawet dysponuje on dokładnymi danymi na temat oferowanych mu systemów, to i tak nie jest w stanie do końca je porównać. Te same rzeczy w ofertach przeważnie nazywają się inaczej, poszczególne pakiety mają inny zasięg tematyczny, różnią się też implementacją pewnych, przeważnie niepotrzebnych zwykłemu użytkownikowi, nowinek technicznych i programowych.

Najbardziej błyskotliwe chwyty reklamowe – testy szybkości, wydajności i efektywności – są dokonywane za pomocą nieporównywalnych konfiguracji sprzętowych i akurat nie na tych, którymi dysponuje użytkownik. Rezultaty testów niewiele też mówią mu o kompleksowej jakości oferowanego oprogramowania i jego dostosowaniu do indywidualnych potrzeb konkretnego systemu informatycznego.

Inną formą współzawodnictwa firm o nasze (tj. użytkowników) kieszenie są mecze produktów programowych. Polegają one na tworzeniu oprogramowania wg zadanej specyfikacji przez zespoły programistów w wyznaczonym i ograniczonym czasie. Oprogramowanie to jest potem oceniane przez zespół ekspertów. Jest to jedna z bardziej obiektywnych form oceny jakości oprogramowania. Częściowo uwzględnia ona ocenę tak ważnych elementów, jak: jakość dokumentacji, stosunek firmy do klientów, stopień rozpowszechnienia itp. Zwycięzcy meczy publikują ten fakt w swoich materiałach reklamowych, natomiast zwyciężeni bagatelizują mecz, mówiąc o złym doborze zadań, słabym przygotowaniu programistów (bo przyjechali akurat z najnowszą wersją oprogramowania i nie mieli czasu na zdobycie doświadczeń,...), niedostosowanej bazie sprzętowej itp. Pozostaje jeszcze problem jakości oprogramowania tych firm, które nie brały udziału w meczu (z różnych zresztą powodów).

Sytuacja taka zaistniała np. w czasie meczu baz danych na IV Górskiej Szkole Polskiego Towarzystwa Informatycznego w Szczyrku w 1992 r. Wzięły w nim udział tylko trzy (z planowanych pięciu) systemy zarządzania relacyjnymi bazami danych (SZRBD): *GUPTA, INFORMIX* i *PROGRESS*. Rezultaty meczu nie są jednoznaczne (tab.). Pomimo niewielkiej wygranej PROGRESS-a w łącznej punktacji, szczegółowa analiza ocen poszczególnych parametrów nie daje już tak jednoznacznej odpowiedzi. Analogicznie, nie wiadomo jak ukształtowałyby się rezultaty meczu, gdyby wzięły w nim udział inne planowane SZRBD: *ORACLE* i *INGRES*. Niezależnie od przyczyn nieobecności firm-dystrybutorów ww. SZRBD, ich brak udziału w tym lub innym meczu nie może dyskwalifikować sprzedawanych przez nich produktów. Utrudnia on tylko ocenę i wybór przyszłemu klientowi tych SZRBD.

Rezultaty meczu baz danych na IV Górskiej Szkole PTI w Szczyrku w 1992 r.

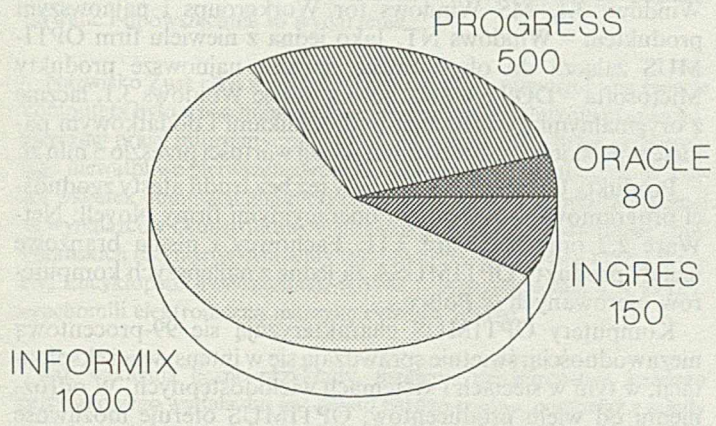
Oceniane parametry SZRBD	Punktacja		
	GUPTA	INFORMIX	PROGRESS
Stopień realizacji zadań	15	33	42
Narzędziowość bazy danych	44	26	31
Elegancja realizacji zadań	42	20	36
Łączna liczba punktów	101	79	109

Źródło: notatki własne

Użytkownicy SZRBD są niekiedy ankietowani przez różne niezależne firmy pod kątem oceny eksploatowanych przez nich produktów. Rezultaty badań ankietowych mają te same wady i zalety, co rezultaty meczy oprogramowania.Przykładowo wg DATA PRO system *PROGRESS* po raz któryś (bodajże piąty) z rzędu został najlepiej oceniony przez użytkowników SZRBD. Ocena ta jest oczywiście oceną łączną, a więc ukierunkowaną na przeciętnego użytkownika baz danych. W zależności od niektórych specyficznych wymagań ocena taka może i wręcz powinna być weryfikowana.

Sposób przeprowadzenia kompleksowej oceny i porównania różnych SZRBD pod kątem własnych potrzeb był głównym

tematem publikacji w kwietniowym numerze *Informatyki* „Systemy zarządzania relacyjnymi bazami danych oraz ich przydatność do budowy informatycznych systemów zarządzania" [1]. Ilustracją metody było pobieżne (nie mające nic wspólnego z obszernymi materiałami reklamowymi firm) przedstawienie i porównanie za pomocą proponowanej przez autorów metody czterech nowoczesnych SZRBD: *ORACLE, PROGRESS, INFORMIX* i *INGRES*. Porównanie zostało przeprowadzone z uwzględnieniem 13 (nomen omen?) cech systemów. Dla każdej cechy zostały przyjęte przez autorów artykułu odpowiednie wartości wag. Wagi te uwzględniają „ważność" poszczególnej cechy w kompleksowej ocenie jakości oprogramowania. Zarówno zbiór cech, jak i ich wagi są elementem subiektywnym, co zresztą zostało wyraźnie zaznaczone w artykule. Subiektywność tych elementów metodyki oceny i porównania SZRBD jest czynnikiem umożliwiającym przeprowadzenie takich analiz dla każdego konkretnego przypadku. Umożliwia ona uwzględnienie nawet najbardziej specyficznych wymagań przyszłego użytkownika systemu.



Szacunkowa liczba licencji SZRBD w Polsce

Autorzy wspomnianego artykułu spodziewali się polemiki co do zakresu i ważności cech, analiz jakości i porównań SZRBD pod kątem potrzeb konkretnych użytkowników, opracowania i publikacji innych metod kompleksowej oceny jakości oprogramowania. Słowem: konstruktywnej dyskusji, służącej przede wszystkim przyszłym użytkownikom SZRBD.

Pierwszym odzewem (poza licznymi telefonami z prośbą o udostępnienie bardziej szczegółowych analiz i opracowań) była publikacja w sierpniowym numerze *Informatyki* tekstu D. Kosmowskiej-Miszalskiej i P. Trojanowskiego *INFORMIX – fikcje i fakty* [2]. Zarówno zawartość tekstu, jak i jego forma zmusiły mnie do tej polemiki. Tekst ten składa się wyraźnie z dwóch części:

- pierwszej, nawiązującej do publikacji [1] w *Informatyce* i będącej próbą jej dyskredytacji w niezbyt poprawnej, moim zdaniem, formie; taką formą jest np. pozbawione jakiegokolwiek uzasadnienia w tekście stwierdzenie: *Próby takie* (tj. porównania SZRBD – przyp. aut.) *pozostawiały zwykle uczucie niedosytu, częstokroć zdziwienia uzyskanymi wynikami oraz zażenowania przedstawionymi wnioskami...*;
- drugiej, zajmującej ponad 90% tekstu i będącej materiałem reklamowym najnowszej wersji *INFORMIX-a* (wersji 5,0 lub też 5,01) oraz przedstawiającej (niekiedy w bardzo skrótowej, bo jednozdaniowej postaci) różne produkty INFORMIX-a.

W części pierwszej brak jest konkretnych zarzutów postawionych metodyce czy też sposobowi doboru cech i ich wag. Podkreśla się natomiast parokrotnie fakt wykorzystania przez autorów opisów różnych wersji SZRBD.

SZRBD są jedną z najbardziej dynamicznych dziedzin oprogramowania i nie jest możliwe utrafienie w moment czasu, kiedy wszystkie cztery rozpatrywane w [1] firmy już wprowadziły na rynek swoje najnowsze produkty, a nie np. zapowiadają wprowadzenie na następny miesiąc lub też już zaczęły sprzedaż, ale w Stanach Zjednoczonych. Pomijając ten fakt, chcę na zasadzie usprawiedliwienia wyjaśnić, że jednym z materiałów źródłowych omawianego materiału było wydawnictwo PTI, które zawiera opisy wszystkich czterech rozpatrywanych SZRBD [3]. Wśród autorów tych materiałów [3] była Pani Danuta Kosmowska-Miszalska, która opisywała *INFORMIX-a*. Nie jest naszą winą, że do druku w tak szacownym wydawnictwie w lecie 1992 trafił opis „wersji dawno wycofanej ze sprzedaży [2]".

Materiały reklamowe *INFORMIX-a*, zawarte w drugiej części tekstu [2], są znacznie obszerniejsze niż opis przedstawiony w artykule [1]. Można jednak mieć do nich sporo zastrzeżeń. Już od początku podawane są bardzo wyselekcjonowane informacje. Świadczą one z jednej strony o mnogości platform, na których można użytkować *INFORMIX-a* (choć bardzo eleganckie stwierdzenie o współpracy *INFORMIX-a* z MS-DOS-em „przez połączenia komunikacyjne [2]" nasuwa określone podejrzenia). Z drugiej strony zaś dotyczą rynku oprogramowania, ograniczonego do systemów UNIX.

Dużą część tekstu [2] zajmuje autorom wyliczanie cech *INFORMIX-a*, które charakteryzują w zasadzie wszystkie bez wyjątku nowoczesne SZRBD, a już na pewno omawiane w [1] cztery najbardziej znane (tj. *ORACLE, PROGRESS, INFORMIX* i *INGRES*). Wszystkie SZRBD są bowiem dostosowane do ciągłej pracy, mają mechanizmy archiwizacji danych w trybie on-line, wysoki stopień zabezpieczenia danych oraz dużą niezawodność działania. Nie wspomnę o takich „drobiazgach", jak rozproszoność bazy danych i przetwarzań, język 4GL i system poleceń SQL, przenoszalność, skalowalność, architektura klient-serwer, dwufazowy protokół potwierdzeń, nieograniczoność (lub praktyczna nieograniczoność) logiczna wielkości bazy danych, możliwości monitorowania i strojenia serwera bazy danych, a także innych cechach reklamujących *INFORMIX-a*. Pewne elementy, wprowadzone do najnowszej wersji *INFORMIX-a* i zachwalane w tekście [2] jako wyjątkowe osiągnięcia, znane są z innych (niekiedy wcale nie najnowszych wersji) SZRBD. Tak ma się sprawa z określaniem więzów integralności bazy danych (zawarte są one w bodajże wszystkich wersjach *PROGRESS-a* i *INGRES-a*), zarządzaniem dużymi danymi binarnymi (*INGRES*), implementacja programowania sterowanego zdarzeniami (*PROGRESS v.* 7) itd.

Pewne zainteresowanie wzbudziła informacja o możliwościach zaoszczędzenia kosztów i zakupie *INFORMIX-a* w wersji *Runtime*. Bliższe zapoznanie się z tym elementem reklamy *INFORMIX-a* pozostawiło wiele pytań, w tym i najważniejsze z nich: dlaczego wersja „tylko do uruchomienia" kosztuje aż 30–70% ceny wersji pełnej (w innych SZRBD cena *Runtimu* wynosi poniżej 10%)?

Reasumując: uważam, że INFORMIX, podobnie jak i inne wspomniane wyżej SZRBD, nie jest złym produktem programowym. Nie jest on oczywiście pozbawiony jakichkolwiek wad, bo ma je każdy system.

Wyjaśniam także, że artykuł [1] nie miał na celu dyskredytację tego lub innego systemu. Miał on pomóc przyszłemu użytkownikowi w wyborze SZRBD, dając mu stosunkowo proste narzędzie do oceny i porównania systemów stosownie do swoich indywidualnych potrzeb. Dlatego też w ślad za autorami tekstu [2] wzywam przyszłych użytkowników do przeprowadzenia „własnych rzetelnych analiz porównawczych [2]", nie obrażając się za sugerowaną w [2] stronniczość czy też nawet nierzetelność.

dokończenie na s. 24

Fenomen OPTIMUSA

W latach sześćdziesiątych (a więc w okresie zastoju polskiej gospodarki) spontanicznie powstał w Polsce znaczący przemysł komputerowy, którego własne konstrukcje pod względem nowoczesności i cech użytkowych znacznie wyprzedzały to, co produkowano w krajach RWPG. Analogicznie w ciągu ostatniego pięciolecia (a więc także w okresie załamania gospodarki) pojawił się w tej samej dziedzinie przemysłu zaskakujący fenomen rozwoju gospodarczego. Oto jeden z wielu powstałych w tym okresie producentów komputerów osobistych klasy IBM PC (a właściwie montowni), w dodatku zlokalizowany nie w tradycyjnych ośrodkach przemysłu komputerowego, jakimi od przeszło ćwierćwiecza były Warszawa i Wrocław, osiągnął w tym roku nie tylko pozycję lidera branży (patrz ranking firm komputerowych opublikowany 15.11 br. przez *Życie Warszawy*), ale również znaczącego producenta w skali europejskiej. Producentem tym jest firma OPTIMUS z Nowego Sącza, której historia powstania, aktualne osiągnięcia i perspektywy rozwoju są warte odnotowania również w INFORMATYCE, rejestrującej już od prawie 30 lat istotne fakty rozwoju techniki komputerowej w kraju i na świecie.

Początki były skromne. OPTIMUS w 1988 r. zaczął od montażu komputerów osobistych z importowanych podzespołów i sprzedawał je na rynku krajowym. Tym co różniło OPTIMUSA od innych krajowych firm komputerowych, było położenie głównego akcentu na jakość sprzedawanego wyrobu. Wynikało to z przyjęcia zasady, że najważniejsze podzespoły będą sprowadzane tylko od producentów markowych, a nie – najtańszych.

Drugm fundamentem przyszłego sukcesu firmy było twarde postanowienie jej twórcy i obecnego dyrektora, Romana Kluski (39 lat!), iż zyski będą w całości inwestowane w rozwój przedsiębiorstwa. W konsekwencji przyjęcia takich zasad OPTIMUS szybko przekształcił się z niewielkiego warsztatu w sporą montownię, a następnie w nowoczesny zakład produkcyjny. Komputery ze stylizowanym niebieskim napisem OPTIMUS szybko wyróżniły się spośród masy tzw. składaków, a nazwa firmy zaczęła być synonimem jakości i niezawodności. Renoma firmy, wzrastający popyt na jej wyroby oraz sprawne zarządzanie sprawiły, że OPTIMUS nie miał kłopotów z uzyskaniem kredytów, co pozwoliło na rozbudowę przedsiębiorstwa, rozszerzenie oferty i stałe podnoszenie jakości, a także niezbędną rozbudowę sieci sprzedaży i serwisu.

W swoich komputerach OPTIMUS zastosował produkty najwyższej światowej jakości: dyski twarde Maxtora, sterowniki Adapteca, napędy dysków elastycznych i monitory NEC-a. O uznaniu dla jakości produktów OPTIMUSA świadczy to, że wspomniane firmy mianowały go swoim oficjalnym dystrybutorem na Polskę. Firmy te nie zawiodły się na polskim partnerze i np. w ub. roku Maxtor przyznał OPTIMUSOWI specjalny dyplom za najszybszy wzrost sprzedaży swych wyrobów w Europie.

Innym znaczącym sukcesem OPTIMUSA była zgoda japońskiej firmy Seikosha na oferowanie jej znakomitych drukarek pod jego własną nazwą. Warto podkreślić, że jest to jedyny tego typu wypadek w Polsce.

Wreszcie w br. OPTIMUS odniósł kolejny, prawie niewiarygodny sukces – firma IBM zawarła z nim porozumienie partnerskie, na mocy którego rozpoczęto produkcję komputerów opartych na oryginalnych podzespołach IBM. Tym samym OPTIMUS został uznany przez Big Blue za producenta OEM (*Original Equipment Manufacturer*). Co ciekawsze, propozycja wyszła od IBM.

Niedawno OPTIMUS zawarł następne porozumienie typu OEM ze znaną amerykańską firmą WYSE Technology, m.in. na montaż systemów wieloprocesorowych z podzespołów tej firmy.

Jakość komputerów z Nowego Sącza sprawiła, że firma nie miała żadnych trudności z uzyskaniem certyfikatu zgodności ze środowiskiem operacyjnym firmy Microsoft Corporation, które w ostatnich latach stało się w istocie światowym standardem. Dzisiaj sprzęt OPTIMUSA ma atesty pełnej zgodności z MS Windows 3.1, MS Windows for Workgroups i najnowszym produktem – Windows NT. Jako jedna z niewielu firm OPTIMUS załącza do oferowanego sprzętu najnowsze produkty Microsofta – DOS 6.0 oraz polską wersję Windows 3.1, łącznie z oryginalnymi dyskietkami, podręcznikami i dodatkowym pakietem polskich fontów o sumarycznej wartości przeszło 5 mln zł.

Produkty OPTIMUSA zdobyły też bez trudu atesty zgodności programowej z systemami operacyjnymi firmy Novell: NetWare 2.2 oraz NetWare 3.11. Fachowcy i pisma branżowe uznały produkty OPTIMUSA za jedne z najlepszych komputerów oferowanych w Polsce.

Komputery OPTIMUS charakteryzują się 99-procentową niezawodnością, świetnie sprawdzają się w intensywnej eksploatacji, w tym w sieciach i systemach wielodostępnych. W odróżnieniu od wielu producentów, OPTIMUS oferuje możliwość rozbudowy obecnie oferowanych komputerów do klasy 486DX3-99, a także jest przygotowany do zastosowania najnowszego procesora firmy Intel - Pentium. OPTIMUS na wszystkie swoje komputery oferuje dwuletnią gwarancję.

OPTIMUS jest obecnie największym holdingiem komputerowym w Polsce. Jego obroty w 1992 r. wyniosły 903 mld zł. W tym samym roku firma sprzedała ok. 50 tys. komputerów w kraju i za granicą, co sprawiło, że w rankingu brytyjskiego czasopisma PC Europe (nr 2,1993) zajęła 11. miejsce wśród europejskich producentów komputerów. Sprzedaż tegoroczna, szacowana na ok. 100 tys. powinna uplasować firmę na piątym miejscu. Na rok przyszły OPTIMUS planuje sprzedaż 200 tys. komputerów, w tym przeszło połowę na eksport. W obecnej chwili firma ma 28 oddziałów, w tym cztery zagraniczne (Holandia, Słowacja, Białoruś i Ukraina), a także dealera w Czechach. Sieć sprzedaży tworzy ok. 300 firm dealerskich. W kraju sprzęt OPTIMUSA można kupić w 230 sklepach, gdzie ceny wszystkich produktów firmy są identyczne, zgodnie z obowiązującą w holdingu polityką cen.

OPTIMUS zatrudnia obecnie łącznie 450 osób.

Przypadek OPTIMUSA jest dobitnym przykładem, jak fałszywe są poglądy niektórych ekspertów gospodarczych, a w ślad za nimi wielu polityków, na temat niemożliwości szybkich przekształceń gospodarczych oraz niezbędności wzmocnienia protekcjonizmu państwowego. Poglądy takie uległy ostatnio wzmocnieniu, mimo że życie dowiodło o bezskuteczności podtrzymywania czy ratowania dotychczasowych struktur (casus ELWRO!). Jeśli ta prawda dotrze do świadomości osób odpowiedzialnych za rozwój gospodarki, to można mieć prawie pewność, że niektóre predyspozycje Polaków (czego klasycznym dowodem jest fenomen rozwoju OPTIMUSA) pchną nasz kraj na ścieżkę przyspieszonego rozwoju, a nawet stworzą realne przesłanki, aby stał się on wzorcem takiego rozwoju w naszym regionie świata.

Konrad Zuse
Profesorem Honorowym Uniwersytetu Szczecińskiego

Prof. Konrad Zuse (z lewej) przyjmuje nominację

Doniosłe historyczne wydarzenie, jakim było uroczyste nadanie przez Uniwersytet Szczeciński w dniu 11 maja br. tytułu profesora honorowego niemieckiemu wynalazcy komputera, Konradowi Zuse, nie została niestety odnotowana przez większość krajowych środków masowego przekazu. Dzięki uprzejmości organizatorów, przedstawiciel INFORMATYKI miał możliwość uczestniczenia w ceremonii, co pozwala utrwalić i upowszechnić to wydarzenie.

Nazwisko Zuse jako wynalazcy i konstruktora pierwszego na świecie komputera było u nas znane jedynie nielicznym czytelnikom niemieckojęzycznej prasy informatycznej, która od co najmniej 30 lat eksponuje jego niewątpliwe pierwszeństwo w skonstruowaniu komputera. W Polsce wskutek dominacji anglojęzycznej prasy i literatury informatycznej za wynalazców i konstruktorów pierwszego komputera uważa się amerykańskich inżynierów Mauchly'ego i Eckerta, którzy (wg naszej Wielkiej Encyklopedii Powszechnej) w 1942 r. zaprojektowali, a w 1947 r. uruchomili elektroniczną maszynę cyfrową ENIAC.

Jak wskazują liczne przykłady, ustalenie pierwszeństwa każdego przełomowego wynalazku jest często bardzo trudne, ponieważ z reguły wynalazki takie są dokonywane niezależnie i prawie równocześnie w różnych krajach, co oczywiście stanowi powód do rozbudzenia nacjonalistycznych ambicji. W przypadku komputera historycznie udokumentowane fakty potwierdzają niewątpliwe pierwszeństwo Zusego: niedoskonałe jeszcze w działaniu, ale z wykorzystaniem systemu dwójkowego, prototypy maszyn cyfrowych o konstrukcji przekaźnikowej zbudował on w 1938 r. (Z-1) oraz 1939 r. (Z-2), natomiast model wykorzystywany już do użytkowych obliczeń w 1941 r. (Z-3). Chronologia tych faktów rozwiewa wątpliwości co do pierwszeństwa niemieckiego konstruktora. Na korzyść Zusego dodatkowo świadczy fakt, że wspomniane prototypy zbudował całkowicie z prywatnych środków i dopiero po założeniu w 1940 r. własnej firmy zaczął realizować zamówienia ze środków zewnętrznych. Sześcioletni okres, jaki dzieli Z-3 od ENIAC-a wskutek szybkiego postępu technologii elektronowej sprawił, że amerykańska konstrukcja miała wprawdzie niewspółmiernie większą moc obliczeniową, ale jej twórcy od samego początku korzystali ze znacznych subwencji. Przy okazji warto przypomnieć, że również Anglicy starają się pretendować do pierwszeństwa wynalazku, ale wspomniane fakty z chronologii osiągnięć niemieckiego konstruktora są trudne do podważenia.

Wykład prof. Konrada Zuse

W 1949 r. Konrad Zuse założył firmę, która aż do 1966 r. produkowała seryjnie stale udoskonalane komputery serii Z (aż do Z-25). Zaostrzająca się konkurencja ze strony wielkich koncernów elektronicznych doprowadziła do likwidacji firmy i jej włączenia do koncernu SIEMENS-SA. Po wycofaniu się z działalności gospodarczej Zuse poświęcił się pracy naukowej i dopiero wtedy został odkryty jako pionier technologii komputerowej. Udekorowano go najwyższymi odznaczeniami państwowymi, nadano tytuły doktorów honoris causa i profesorów honorowych wielu uczelni o światowej renomie, jego nazwiskiem nazwano liczne szkoły i ulice. Inicjatywa Uniwersytetu Szczecińskiego była właśnie jednym z dowodów uznania wielkich zasług prof. K. Zusego.

Nadanie Konradowi Zuse tytułu profesora honorowego odbyło się w tradycyjnej uroczystej oprawie. Ze strony władz uczelni wystąpił JM Rektor – prof. Tadeusz Wierzbicki, Dziekan Wydziału Ekonomicznego – prof. Antoni Nowakowski oraz Dyrektor Instytutu Cybernetyki Ekonomicznej i Informatyki – prof. Wojciech Olejniczak. W ceremonii uczestniczyli również goście niemieccy: prof. Hermann Flessner, przyjaciel i długoletni współpracownik Zusego, który wygłosił referat przybliżający sylwetkę wielkiego wynalazcy, prof. Peter Reusch, współpracujący od kilku lat z Uniwersytetem Szczecińskim, który naświetlił dotychczasowy i przyszły rozwój międzynarodowej współpracy Instytutu CE i I, oraz prezes Niemieckiego Towarzystwa Informatycznego – prof. Roland Vollmar i Konsul Generalny RFN w Szczecinie Reinold Roesch, którzy wygłosili okolicznościowe przemówienia. Bezpośrednio po wręczeniu aktu nominacyjnego prof. Zuse, mimo ukończonych 83 lat nadal całkowicie sprawny, wygłosił w pełni improwizowany z pamięci i znakomity swą klarownością wykład na temat trudnej, ale podniecającej ciągłej walki każdego wynalazcy o realizację swych pomysłów. W podsumowaniu wykładu Profesor skierował apel do licznie zgromadzonych w auli studentów o nieustanne twórcze poszukiwania i stawianie sobie – podobnie jak każdy wynalazca – ambitnych zadań. Miłym zakończeniem ceremonii było wystąpienie przedstawicieli środowiska studenckiego wygłoszone w języku niemieckim i połączone z wręczeniem Profesorowi kwiatów.

Redakcja INFORMATYKI pragnie wyrazić Władzom uczelni a także dr. hab. Zygmuntowi Drążkowi, który walnie przyczynił się do uzyskania zgody i przyjazdu do Szczecina prof. Zuse, wielkie uznanie za pionierską inicjatywę uhonorowania twórcy komputera oraz przyłącza się do życzeń złożonych Profesorowi Honorowemu przez uczestników uroczystości.

WK

Quo vadis...

dokończenie ze s. 21

Tym bardziej, że sugestie te pochodzą od dystrybutorów INFORMIX-a, który nie wypadł najlepiej w ocenie podanej w [1].

Jeśli artykuł [1] na zasadzie „kija w mrowisko" przyczyni się do wzrostu aktywności publikacyjnej (mam nadzieję, że nie tylko reklamowej) dystrybutorów i obecnych użytkowników oprogramowania, to wygrywa na tym przyszły użytkownik, uzyskując zwiększoną dawkę informacji o interesujących go SZRBD.

* * *

Zamiast podsumowania chcę przedstawić szacunkowe dane, dotyczące liczby licencji w Polsce wspomnianych w niniejszym tekście SZRBD (rys.) i zaapelować do dystrybutorów o ich ewentualną korektę. Mam nadzieję, że nie będą oni zasłaniać się tajemnicą handlową. Interpretacja danych z rysunku powinna oczywiście uwzględniać także inne elementy, takie jak np. czynnik czasu (liczbę lat obecności danego projektu na polskim rynku), ale o tym może innym razem.

Dyskusja na łamach *Informatyki* dotyczy zatem około 2000 obecnych i prawdopodobnie około 2000 przyszłych użytkowników. Nie są to duże liczby w porównaniu z np. liczbą użytkowników *dBase, CA-Clipper* lub też mniej popularnego *Fox-Base*.

LITERATURA

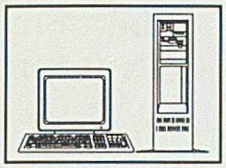
[1] Korbel L., Miłosz E., Miłosz M.: Systemy zarządzania relacyjnymi bazami danych oraz ich przydatność do budowy informatycznych systemów zarządzania. Informatyka, 4, 1993, s. 8

[2] Kosmowska-Miszalska D., Trojanowski P.: INFORMIX – fikcje i fakty. Informatyka, 8, 1993, s. 5

[3] Systemy relacyjnych baz danych: INFORMIX, INGRES, ORACLE, PROGRESS. Materiały trzeciej i czwartej Górskiej Szkoły PTI, Szczyrk, 1992.

Woźniak P.: SuperMemo jako nowe narzędzie wzrostu wydajności programisty na przykładzie programowania w Object Windows INFORMATYKA 1993, nr 12, s. 1 Efekty zastosowania metody SuperMemo do nauki programowania na przykładzie pakietu Object Windows firmy Borland.	**Woźniak P.: SuperMemo as a new tool increasing the productivity of a programmer. A case study: programming in Object Windows** INFORMATYKA 1993, nr 12, p. 1 Effects of the SuperMemo method application for programming learning on example of Borland's Object Windows package.	**Woźniak P.: SuperMemo als neues Hilfsmittel zur Leistungssteigerung eines Programmierers. Ein Musterbeispiel: Programmieren in Object Windows** INFORMATYKA 1993, nr 12, S. 1 Effekte von Anwendung der SuperMemoMethode zur Programmierungsausbildung an Hand von Beispiel des Borland's Object Windows-Pakets.
Subieta K.: Logika teoretycznych urojeń INFORMATYKA 1993, nr 12, s. 5 Krytyczna ocena niewłaściwego stosowania matematyki w informatyce.	**Subieta K.: Logic of theoretic imaginations** INFORMATYKA 1993, nr 12, p. 5 Critical evaluation of mathematics wrong using in computer science.	**Subieta K.: Logik der theoretischen Wahnvorstellung** INFORMATYKA 1993, nr 12, S. 5 Eine kritische Beurteilung der falschen Mathematikverwendung in der Informatik.
Gwiazda J.: Ośrodek obliczeniowy firmy ubezpieczeniowej w Kanadzie INFORMATYKA 1993, nr 12, s. 12 Charakterystyka rozwiązań organizacyjnych i technicznych przetwarzania danych w kanadyjskim towarzystwie ubezpieczeniowym WCB.	**Gwiazda J.: Computing center of an assurance company in Canada** INFORMATYKA 1993, nr 12, p. 12 Characteristics of data processing organizational and technical solutions in WCB – a canadian assurance company.	**Gwiazda J.: Rechenzentrum einer Versicherungsgesellschaft in Kanada** INFORMATYKA 1993, nr 12, S. 12 Eine Charakteristik von organisatorischen und technischen Lösungen der Datenverarbeitung in WCB – einer kanadischen Versicherungsgesellschaft.
Kędzierski S.: Modelowanie obiektów dynamicznych INFORMATYKA 1993, nr 12, s. 16 Charakterystyka podstawowych problemów w obiektowo zorientowanym konceptualnym modelowaniu systemów informatycznych.	**Kędzierski S.: Modeling of dynamic objects** INFORMATYKA 1993, nr 12, p. 16 Characteristics of fundamental problems connected with object-oriented conceptual modeling of data processing systems.	**Kędzierski S.: Modelieren von dynamischen Objekten** INFORMATYKA 1993, nr 12, S. 16 Eine Charakteristik von Grundproblemen in objektorientiertem Konzeptualmodelieren der EDV-Systeme.
Miłosz M.: Quo vadis?, czyli ciąg dalszy o wyborze relacyjnej fazy danych INFORMATYKA 1993, nr 12, s. 20 Krytyka artykułu eksponującego zalety systemu zarządzania relacyjną bazą danych INFORMIX.	**Miłosz M.: Quo vadis, or continuation of relational database choosing** INFORMATYKA 1993, nr 12, p. 20 Criticism of the article which exposes advantages of the INFORMIX relational database management system.	**Miłosz M.: Quo vadis, oder die Fortsetzung von Wahl einer Relationsdatenbank** INFORMATYKA 1993, nr 12, S. 20 Eine Kritik des Artikels, der die Vorteile des INFORMIX-Datenbankverwaltungssystems exponiert.

JUNISOFTEX®



Przedsiębiorstwo Produkcyjno-Usługowe Innowacji Technicznych Biuro Techniczno-Handlowe 44-100 Gliwice ul. Konstytucj 11.
tel. 31-57-00, 31-51-31, 31-75-10, fax 31-75-10, centr. 31-90-81 do 88, telex 036233
Zespół wdrożeń i analiz systemów komputerowych wewn. 15 ,25, 282, Zespół inżynierii oprogramowania wewn. 14, 20, 245, Zespół d/s serwisu systemów Novell i sprzętu komputerowego wewn. 16, 245, Dział Handlowy wewn. 18, 28, 201, Szkoła Komputerowa wewn. 282, Sekretariat wewn. 250, Prezes wewn. 13, 272, Dyrektor wewn. 12, 272, Główny Księgowy wewn. 256,
** Konto: PKO I/O Gliwice 27661-225298-136 **

OFERUJEMY

Kompleksową komputeryzację przedsiębiorstw obejmująca dostawę i wdrożenie zintegrowanych systemów komputerowych takich jak system:

finansowo-kosztowy (automatyczne rozliczanie kosztów)
+
gospodarka materiałowa
+
środki trwałe
+
wyposażenie
+
obrót towarowy fakturowanie sprzedaży
+
kadry-płace
+
techniczne przygotowanie produkcji

Sprzęt komputerowy w tym:

Komputery
Monitory
Dyski twarde
Drukarki (mozaikowe i laserowe)
osprzęt sieciowy (karty sieciowe, kable zwykłe i światłowodowe)

Systemy i oprogramowanie licencjonowane

DOS
Novell
Btrieve
SQL Base
Windows

Szkolenia komputerowe

podstawy obsługi komputerów
arkusze kalkulacyjne (Qpro, Lotus, EXCEL)
edytory tekstów (WordPerfect, TAG, ChiWriter)
bazy danych (dbase IV, Clipper, Paradox)
systemy operacyjne (DOS, Novell)

SOFTWARE SUPPORT FOR DOS, NOVELL 2.2, 3.11, 4.0, SQL BASE firmy GUPTA

*

OPROGRAMOWANIE DLA PRZEDSIĘBIORSTW

oferujemy własnej produkcji zakładowy system informatyczny PHU-Perfect(N) obejmujący swoim zakresem:
system finansowo-kosztowy umożliwiający automatyczne rozliczanie kosztów, atomatyczną dekretację, analizę rozrachunków
system obrotu materiałami, wyrobami, towarami (kody paskowe) w tym: stany magazynowe, ewidencje zamówień, fakturowanie sprzedaży (VAT) z automatyczną dekretacją rozdzielników kosztów i sprzedaży do systemu finansowo-kosztowego,
system ewidencji majątku trwałego i wyposażenia
system kadrowo-płacowy (SQL) z automatyczną dekretacją rozdzielników płacowych do systemu finansowo-kosztowego
system technicznego przygotowania produkcji (uk. 1993 r)
(oprogramowanie w systemie SQL BASE w przygotowaniu)

*

OPROGRAMOWANIE DLA URZĘDÓW MIAST I GMIN

oferujemy własnej produkcji system finansowo-podatkowy UMG-Perfect(N) obejmujący swoim zakresem:
system finansowo-księgowy,
systemy podatkowe obejmujące: ustalanie wymiarów, naliczanie i kontrolę płatności należności podatkowych dla dowolnych podatników i/lub płatników podatków na ich kontach i kontach dochodów ze swobodną korektą dowolnych informacji w dowolnym okresie a w tym:
ewidencja zapłat z wykorzystaniem kodów paskowych, emisja wezwań do zapłaty, naliczanie odsetek.
system ewidencji majątku

*

Zapraszamy do współpracy w zakresie wdrożeń naszych systemów partnerów, - firmy informatyczne z całego kraju !!!

0/7/92

POLSKIE LITERY + NOWE MOŻLIWOŚCI

AKREM (90-950 Łódź 1, box 308, tel. 36-48-74) oferuje pakiety programowe:

POLY do CLIPPERA 87/5

- indeksowanie wg polskiego alfabetu w dowolnym standardzie,
- wprowadzanie, kontrola i manipulacje na danych o polskojęzycznym obrazie (GET-y, funkcje UPPER, ISALPHA itd.),
- praktycznie wszystkie znane metody programowego definiowania i symulacji polskich liter:
  - symulacja na Herkulesie w trybie znakowym i graficznym,
  - nie znikające polskie litery na EGA, VGA, SVGA we wszystkich trybach (w tym tryby tekstowe 25/28/43/50 wierszy, tryby graficzne, 512 fontów w trybie tekstowym),
  - polskie klawiatury (Alt i/lub dowolnie wybrany "klawisz polskiej litery", możliwość zmiany sposobu wprowadzania "w locie", liczne dodatkowe udogodnienia, jak np. automatyczna zmiana wielkości już wprowadzonych liter, "inteligentny "CapsLock, itd.),
  - na drukarce trzy metody , w tym nowość: nie znikający "download" (automatycznie samodefiniujący się po każdym włączeniu drukarki, działa bez żadnych przeróbek na każdej drukarce emulującej standard Epson lub IBM) oraz druk w trybie graficznym i najprostsza metoda "nakładania",
- dynamiczne dostosowanie programu do dowolnego (nawet własnego) standardu polskich liter,
- komunikacja programów w CLIPPERZE z rezydentami pakietu,
- bezbłędna współpraca z dowolnymi wersjami DOS-u,
- a ponadto system tablica/dysk, tablice wielowymiarowe o zwiększonej pojemności, szybkie operacje na tablicach, system pakowania danych (40-50% oszczędności pamięci dyskowej bez straty szybkości i miejsca w pamięci operacyjnej), opcja szyfrowania danych (przydatna dla użytkowników pracujących w sieci), menu wielopoziomowe, dwuwymiarowe i logiczne (wybór wielu opcji jednocześnie), manipulowanie atrybutami i kolorami ekranu, wyświetlanie dużych tekstów w okienkach, uniwersalne podejście okienkowe, polski kalendarz i liczby słownie, możliwość indeksowania i wyszukiwania danych wg polskiej wymowy, opcja definiowania własnych szablonów formatu, "trójwymiarowe" rozszerzenie funkcji DBEDIT, możliwość pracy z ekranami o zwiększonej liczbie wierszy i kolumn, klawisze F11 i F12 itd.

W skład pakietu wchodzą: biblioteka funkcji, cztery programy rezydentne (nakładki na DOS), edytor własnych fontów dla Herkulesa, VGA i drukarki (Draft i NLQ), programy pomocnicze, pliki rozmaitych fontów, programy przykładowe oraz doskonała dokumentacja (podręcznik oczywiście po polsku), a w przypadku CLIPPERA 5.x dodatkowo plik QUICKPOL umożliwiający polonizację bez zmian w programie źródłowym oraz plik NEWORDER definiujący strukturalno-obiektowy język NewOrder (włączający do CLIPPERA m.in. obiekty klasy MENU).

POLY-C

Zawiera wymienione wyżej programy rezydentne i narzędziowe, podręcznik napisany specjalnie dla C oraz bibliotekę blisko 300 funkcji dostarczanych w postaci źródłowej, przystosowanych do użycia wraz z dowolnym kompilatorem języka C na IBM PC (standard ANSI lub C++, szablony dla Borlanda 3.1), a także możliwych do wykorzystania przez znających C programistów w CLIPPERZE, FOX PRO czy też w PARADOX-ie.

Biblioteka funkcji POLY-C zawiera większość funkcji z biblioteki POLY dla CLIPPERA, a ponadto szereg funkcji napisanych specjalnie dla C. W bibliotece została uwzględniona kompleksowa (ekran, klawiatura, drukarka, manipulacje znakowe, konwersja) wzorowo zorganizowana polonizacja programu, a ponadto takie zagadnienia, jak: dowolne operacje na datach i łańcuchach znaków, pakowanie i szyfrowanie danych, pełnoekranowe wprowadzanie danych (obiekty GET), wyświetlanie dużych tekstów w okienkach, uniwersalne podejście okienkowe, funkcja INTPOLY symulująca operacje BIOS-u w pamięci ekranu, manipulacje ekranowe, drukowanie danych, organizacja menu wielopoziomowych, dwuwymiarowych i logicznych (obiekty MENU i PROMPT, menu strumieniowe), przetwarzanie list, help kontekstowy, język NewOrder, komunikacja programu w C z rezydentami pakietu, itd.

NEW INDEX

Nowy system obsługi baz danych w C. Zawiera obsługę obiektów FIELD, zbiorów DBF i DBT z możliwością pakowania pól bez straty szybkości pracy oraz zbiorów NDX, NTX i NZX (pliki indeksowe najnowszej generacji zaprojektowane specjalnie dla C, dowolnie wiele indeksów różnych rodzajów w jednym zbiorze, możliwość programowego ustawienia wielkości strony indeksowej). Udogodnienia do pracy w sieci. Generator Pól. Przezroczysta baza danych. Podręcznik.

POLY-DOS

Pakiet POLY-DOS umożliwia wszechstronną polonizację (tryb tekstowy i graficzny) oraz m.in. przedefiniowanie klawiatury (makrodefinicje dowolnej długości), definiowanie własnych fontów na ekranie i drukarce, komunikacja rezydentów z programami napisanymi w dowolnym języku, itd. Podręcznik.

EURO-SOFTWARE

Do POLY5 i POLY87 można dołączyć również moduły EXPORT umożliwiające przystosowanie programu do dowolnego alfabetu europejskiego (indeksowanie, wprowadzanie danych, duże i małe litery). Zaleca się zakup EXPORT-u łącznie z pełnym POLY-DOS-em, ponieważ można wtedy całkowicie przedefiniować klawiaturę, ekran i drukarkę.

BALL STAR - MECZ

Piłkarska baza danych na IBM PC zawierająca dowolne tabele i prognozy. Podręcznik.

* * *

A oto ceny pakietów (z podatkiem, ale bez opłaty pocztowej).

POLY 87 (do Clippera 87)	- 1.200 tys. zł
POLY 5 (do Clippera 5.0, 5.01 i 5.2)	- 1.400 tys. zł
POLY 5 + 87	- 1.800 tys. zł
EXPORT 5 lub 87	- 200 tys. zł
EXPORT 5 + 87	- 300 tys. zł
POLY-C (źródła)	- 2.200 tys. zł
NEW INDEX (źródła)	- 2.000 tys. zł
NEW INDEX + POLY C (źródła)	- 3.200 tys. zł
POLY-DOS	- 700 tys. zł
MECZ	- 550 tys. zł

Przy jednoczesnym zakupie POLY-C i do CLIPPERA dodatkowe 500 tys. zł zniżki. Zamówienia przyjmowane są listownie lub telefonicznie. Wysyłka za pobraniem.

Oferujemy również CA-CLIPPER 5.2, CA-dBFast 2.0 (baza danych z kompilatorem pod Windows) i inne oprogramowanie firmy Computer Associates. W tym przypadku obowiązuje zamówienie pisemne wraz z kserokopią dowodu wpłaty na konto

AKREM, PBK SA V/O ŁÓDŹ, nr 374606-11729-136.

0/1/93